CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Comprehending How Merge Conflicts Developed in an Open Source Software Project

Master's Thesis in Software Engineering

MOSES MSAFIRI CHAGAMA

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Comprehending How Merge Conflicts Developed in an Open Source Software Project

MOSES MSAFIRI CHAGAMA

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Comprehending How Merge Conflicts Developed in an Open Source Software Project

MOSES MSAFIRI CHAGAMA

Supervisors: Regina Hebig, Computer Science and Engineering Department
Thorsten Berger, Computer Science and Engineering Department

Examiner: Eric Knauss, Computer Science and Engineering Department

Master's Thesis 2018
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Comprehending How Merge Conflicts Developed in an Open Source Software Project
MOSES MSAFIRI CHAGAMA
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

Studies show that merge conflicts are common in collaborative development, as developers work in a same part of a project code base using a version control system. The merge conflicts occurred when the developers merge their changes from their individual branch into a mainline branch or vice versa. Often, the developers are required to resolve the conflicts manually when merge tools are unable to resolve them automatically. However, the developers face a challenge of deciding which changes they should keep during conflicts resolution. This situation occurs when they do not understand the changes done by others to resolve the conflicts without worrying that their changes will introduce faults. The purpose of the study is to explore merge conflicts and changes that lead to their occurrence in the project code base. To achieve this purpose, we conducted an in-depth manual analysis of conflicting changes in a sample of 40 merge conflicts from an open source software project. The merge conflicts were categorized for comprehension and analyzed for changes that led to their occurrence in the project code base. Six categories of the merge conflicts and eight categories of changes that led to the merge conflicts in the project code base were identified. We observed that most of the merge conflicts are due to changes in method call or object creation statements and most of the changes in the project code base that led to the merge conflicts are related to introduction of new features and refactoring tasks.

Keywords: open source software, merge conflicts, collaborative development.

Acknowledgements

I would like to thank my supervisors Regina Hebig and Thorsten Berger for their guidance and support throughout the thesis work. Also, I would like to thank Wanzi Gu and Okorie Chidiebere Anthony for taking their time to review this thesis. Lastly, I would like to thank Chalmers University of Technology for financial support which made possible for me to pursue my studies including the thesis work.

MOSES MSAFIRI CHAGAMA, Gothenburg, January 2018

Contents

1
Introduction

In collaborative development, version control systems allow developers to use branches and to work individually in a part of software project [1]. The developers can implement and experiment with changes independent of a main development branch or repository. The changes may include bug fixes, introduction of new features or enhancement of existing features, and refactoring. When the developers are done, the changes are then reviewed and integrated (or merged) into the main development branch. However, the changes are integrated into the main development branch if they are deemed sufficient.

The practice of *branching* is common in both closed source software projects [1] and open source software projects [2]. In open source software projects, contributors who have permissions to make changes to project repository can create new branches while those who do not have permissions submit patches or pull requests [3]. Both approaches in the end integrate changes into the main development branch.

Despite use of branches helping developers to work without interruptions [2] and less development time [4], conflicts are common when integrating changes into main development branch [5, 6, 7]. Kasi and Sarma [7] found occurrence of merge conflicts on average 23% in their study. Similarly, Brun et al. [5] found that developers experienced textual merge conflicts in 16% of changes integration. Often, merge tools resolve those conflicts automatically; however, the developers may be required to resolve the conflicts manually if the merge tools were unable to resolve automatically. During the resolution of the merge conflicts, a developer may use one of a version involved in the merge conflict completely, that is, either his or her version or the other conflicting version, or incorporating both versions. To decide which version to use, the developers must understand changes in both versions, especially, the other version so that to make the right decision. In this situation, a challenge arises if the developer does not understand clearly what and why the other changes were made. This challenge was reported in a prestudy that preceded a main study in the thesis work. In the prestudy, it was found that understanding implementation of features during feature integration into the main development branch was a challenge when there are merge conflicts, which required manual resolution. Similar challenge was found in a study by McKee et al. [8] which was published after the prestudy. McKee et al. [8] results show that developers perceive the merge conflicts to be difficulty if they lack knowledge or expertise in the area of the conflicting code. Therefore, given that merge conflicts are prevalent when developers integrate changes into the main development branch or incorporate changes into their individual branches and there

is a challenge of understanding the conflicting changes during conflicts resolution, understanding the merge conflicts and changes that lead to their occurrence is crucial towards aiding developers during conflicts resolution.

The purpose of the study is to understand merge conflicts and to explore changes that lead to merge conflicts when merging changes from a branch such as feature branch into main development branch and vice versa in an open source software project. To achieve the purpose of the study, we qualitatively conduct an in-depth analysis of changes that led to merge conflicts and categorize the merge conflicts using a sample of conflicting changes in the merge conflicts. The following questions guided the work in this thesis:

- **RQ1**. How can the merge conflicts be categorized in terms of code level changes?
- **RQ2**. What are the project level changes that led to the merge conflicts in an open source software project?

This thesis makes the following main contributions:

- Two datasets of conflicting changes in merge conflicts in an open source software project. One dataset is for conflicting changes in the merge conflicts and another is for description of changes led to the merge conflicts in a sample of conflicting changes in the merge conflicts.
- Methodology used to extract and analyze the changes that led to the merge conflicts.

The remainder of the thesis is organized as follows: Chapter 2 contains background on open source software and development, collaborative development models, merge conflicts, and related work. Chapter 3 reports on the prestudy prior to the main study. Chapter 4 describes the methodology used to find and analyze merge conflicts. Then, Chapter 5 presents the results of the study. Chapter 6 discusses the outcome of the study and threat to validity. Finally, Chapter 7 draws conclusion and offers possible further research.

2
Background

This chapter presents background on open source software and its development. It describes collaborative development models used in open source projects. Background on merge conflicts and related work are provided.

2.1 Open Source Software and Development

According to Open Source Initiative (OSI) [1], open source software is software that anyone can use it freely, obtain its source code, modify the source code, and share original or modified source code without paying or asking permission to original developers. Most open source projects share two characteristics: adherence to Open Source Definition (OSD) [9] provided by OSI and often developers are users of the software [10]. The OSI outline 10 criteria that any open source software should follow. The criteria can be summarized into three main aspects: right to modify the software, ability to distribute the software freely, and availability of the source code [10]. Many of the criteria in OSD are based on provision of licenses that do not restrict on redistribution and modification of the software, discrimination against anyone or specialization, and discrimination against other software.

One advantage of open source software is that most developers (contributors) are also users of the software [10, 11, 12]. This help to make the software stable and usable. The contributors help to debug the software and report defects to main project repository or project core developers. This can speed up development process in which core contributors or developers focus on development of new features while other contributors help to test developed features. Also, contributors can fix reported defects. Moreover, as users, contributors in turn help to improve quality of the software they use. Furthermore, contributors can request new features and help to develop them as they may have specific requirements which may also be important to other users. Therefore, contributing those features will eventually benefit the project.

In most cases, open source software is developed by a developer or community of developers who are volunteers. The developers are motivated by several factors to participate as most of them are not paid for their work. First, developers regard contribution to an open source development as a hobby that they are enjoying doing it [13, 14]. The developers feel satisfied, fulfilling, and competent when writing

[1] https://opensource.org/

programs [13]. Second, developers contribute to open source projects to improve their career profiles, for example, improving programming skills [13, 14] and gaining reputation [13, 14]. Therefore, through their contribution, developers are investing in their career development. Third, developers feel obligated to give back to community after they had received help [14]. Fourth, developers are driven by desire to make open source the software they developed as a result of personal needs [13].

Furthermore, developers are driven by need to choose a software which they can change the code to meet their needs [14]. Unlike commercial software, open source software allows users such as organizations to obtain source code which they can changes to fit their needs. Most users feel compelled to share the changes back to the community so that others can benefit with added features [14].

Open source software contributors usually are in different geographical location [11, 15]. This pose some challenges on coordinating development activities, in particular source code management. Nevertheless, version control system in open source software projects allow to keep track of changes from various contributors. However, with distributed nature of open source development, conflicts arise related to source code control [16]. Contribution from two or more developers who make parallel changes on same part of source code can results to conflicts when merging changes to the main project repository.

2.2 Collaborative Development Models

Distributed version control system (DVCS) such as Git[2] allows developers to share code without using a main repository [17]. Unlike centralized version control systems (CVCS) where there is a master central repository and developers share their code through it [18], the DVCS offers flexibility of reviewing and fixing changes before submitting to the main repository. The following development models are used in open source software projects.

2.2.1 Pull-Based Model

Pull-based model is a form of collaborative development in which developers work on a copy (clone) of shared repository, and send changes to a main repository once they are done. The pull-based model is overall gaining popularity in distributed software development, in particular open source software projects, helped by distributed nature of version control systems such as Git [4].

With use of code hosting sites such GitHub[3] which offer pull request feature, anyone who wants to contribute can submit contribution to the main project repository after forking the project, that is, obtaining the copy of the project. The pull request feature allows contributors to submit their contribution and receive feedback for

[2]https://git-scm.com/
[3]https://github.com/

further improvement. However, it should be noted that the pull request feature can also be used in a branching model for code review process before changes are merged into main development branch. Therefore, in this thesis, the pull request process is limited to integrating changes in the main development branch from forks.

The pull-based model is effective in contributing to the open source software projects through use of pull requests compared to traditional patch submission [4]. It takes less time for the pull request to be accepted after submission. Most pull requests are merged within 4 days [4]. Moreover, the pull-based model allows easy inspection process of changes made by contributors.

However, the pull-based model has notable key challenges to both contributors and core team who integrate the contribution in main repository. For contributors, the challenges include: understanding project code base [3], use of collaboration tools such as Git [3, 19] and timely feedback from core team [3]. On the other hand, for integrators of the contribution, the challenges include: maintaining quality of the code and providing feedback to contributors without discouraging them [20]. Nevertheless, tutorials on version control tools such as Git, a good documentation of the source code and project contribution guidelines can reduce most of the challenges for new contributors.

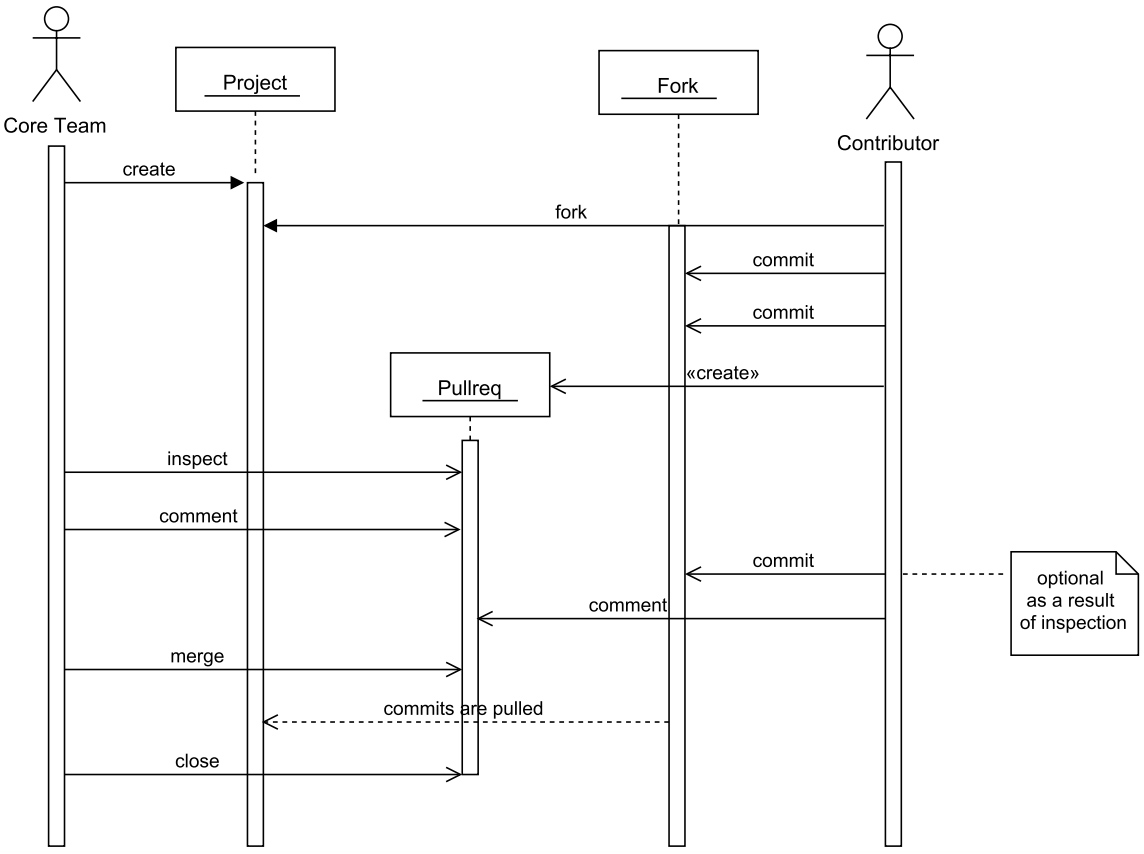

Figure 2.1: The pull request process [4]

A typical pull-based model has two roles [4] namely: core team and contributor. However, during the inspection process, the pull-based model can have three roles [21] namely: a core member who judge the pull request, commenter who comment on the changes in the pull request and contributor. As shown in Figure 2.1, the process of contributing to an open source software project for most contributors starts with forking the main repository.

However, the practice of code forking is not limited only for a purpose of contributing to open source software projects. Other reasons include: discontinuation of main project and experimental purpose when introducing major changes into the project [22, 23]. Most forks have shorter active period apart from those started from the discontinuation of the original project and do not synchronize with a main project after needs were met [24]. Nevertheless, for contributions in open source software projects, forking is the first step to obtain a copy of the main repository.

After forking a project, the contributor implements necessary changes and then creates a pull request, that is a request to merge the changes into main repository, after satisfied with the changes. The contributor can provide information about what the changes accomplish so that to ease inspection process. The core team inspect and comment on the pull request. Depending on the review process and guidelines, further information might be needed. The contributor can then add more changes as part of the review process. Then the pull request is merged into main repository if the core team accept the changes.

However, not all pull requests can be accepted. The pull requests can be rejected because of similar changes in other branches, uninterested changes, quality of the changes, and implementation error [4]. The core team can then close the pull request if the changes are accepted or rejected which is a last step of the pull-based model.

2.2.2 Branching Model

Version control systems offer branching feature which allow developers to work simultaneous in a same part of a system by tracking multiple changes made on the source code [25]. The developers create branches, that is, diverging from a mainline development [26], to implement necessary changes in the project. Most branches are used for release purpose and feature implementation [25]. The mainline branch is used as a central repository same way as in a centralized version control system (CVCS) [18].

Figure 2.2 shows a simple time-line example of a branching model in Git version control system. A first step in the branching model is to create a branch from a *mainline* branch, which is a main repository (or development) branch. In this thesis, the new branch is referred to as a *topic branch* which may be created to implement new feature, enhancing new feature, fixing issue or refactoring part of the code. The new branch points to a commit which is a base between mainline and topic branch. The base commit is referred to as *common ancestor*. The next step after

implementing changes is to *merge* the changes into the mainline branch. When merging topic branch changes into mainline branch a commit is created. The commit is referred to as *merge commit*. The merge commit has more than one parent, that is, it points to more than one commit. However, in Git, integration of changes can also be done using *rebase* [26] where changes from one branch are reapplied onto another branch. In this thesis, merging refers to the basic merge which creates the merge commit.

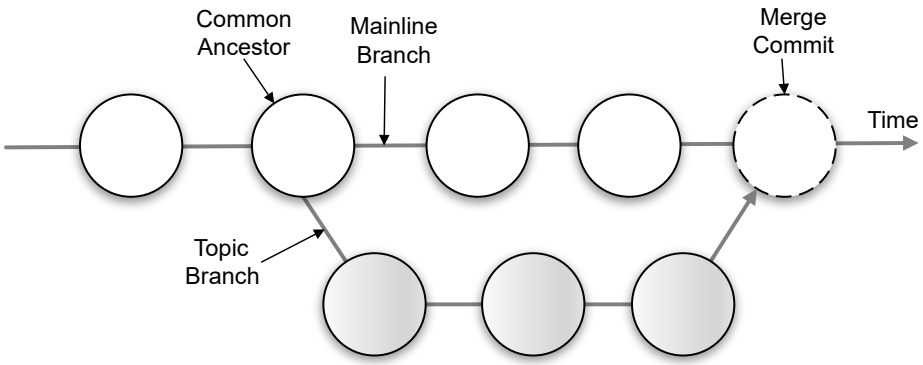

Figure 2.2: A simple example of a branching model

One advantage of the branching model is that it allows support for simultaneous releases [27]. One branch can be created to fix minor issues before a release while in other branches new features for next releases are developed. The releases branches are labeled to reflect features included in a version.

However, contribution in an open source software project using the branching model has a drawback. Contributors need to have permission to make changes in a main repository [4]. This is a challenge for new contributors since they need to build trust with a core team before they are granted permission.

2.3 Merge Conflicts

Merge conflicts are one of the challenges in collaborative development [5, 6, 7]. Developers work on separate branches or local copy of main repository and merge (or integrate) their changes into a project mainline branch in a version control system (VCS) such as Git. Often merging of changes into the mainline branch is done automatic. However, the merging process is done manually when there are conflicts [28]. The merge conflicts occur when there are parallel changes on the same part of the code. In the event of a merge conflict, the version control system indicates conflicting file and part of the code. However, it depends on the version control system on use. In Git, conflicting versions are enclosed in markers. Figure 2.3 shows an example of a merge conflict in Git.

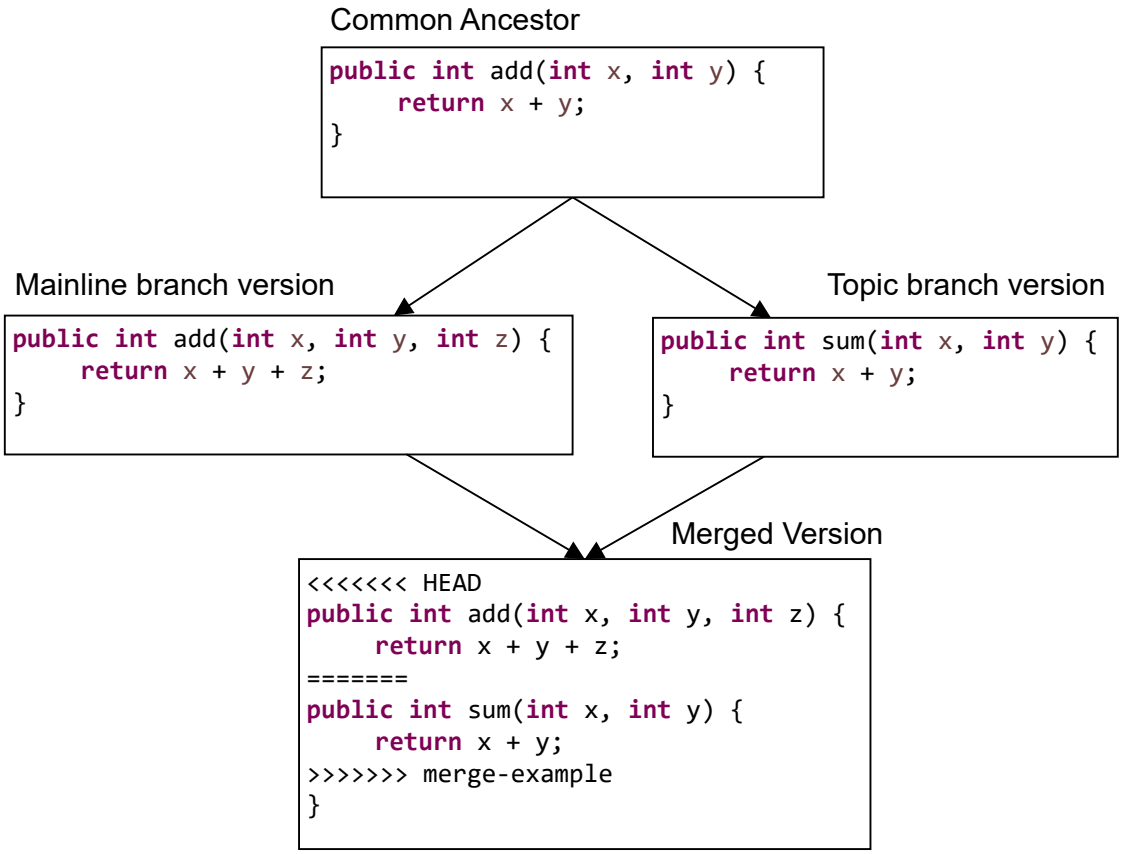

Figure 2.3: An example of a merge conflict in Git

With existences of several merge techniques based on software artifacts [29], merge conflicts can be divided into three kinds of conflicts: textual, syntactic, and semantic conflicts.

Textual conflicts: These conflicts arise in line-based merging [29]. Textual merge tools compare parallel lines of text files for modification, deletion, insertion, and arrangement of text. However, textual merging techniques cannot combine parallel modification on same line [29]. Usually, the textual merge tools report conflicts such as space changes and code comment modification which might not be important when merging changes into mainline branch.

Syntactic conflicts: Unlike textual merging, syntactic merging consider syntax of the source code file during merge [29]. Syntactic merge tools report conflicts when merged changes are not syntactically correct, therefore conflicts such as space changes can be ignored. However, the syntactic merge tools cannot detect changes which result to semantic problems [29] despite are syntactic correct.

Semantic conflicts: These conflicts raise errors such as undeclared variables when merged changes are compiled. The conflicts are referred as static semantic conflicts. They can be detected by graph-based merge approaches [29] which there is a link between declaration or definition and invocation of variables and methods (or

functions).

2.4 Related Work

Most studies of merge conflicts in a collaborative development focus on aiding developers by providing awareness to minimize merge conflicts and its impact. However, to the best our knowledge, there exist few studies that focused on understanding the structure of the conflicting changes in the merge conflicts. Accioly et al. [30] conducted a study of merge conflict characteristics to understand the existing patterns of the merge conflicts and determine the requirements and recommendations to improve available tools used for the merge conflict resolution and detection. In their study, they analyzed the merge conflicts in a sample of 123 open source Java projects by replaying merge scenarios using a semistructured merge tool called SSMerge [31]. The semistructured merge tool takes advantages of both unstructured and structured merges. Accioly et al. [30] found nine conflict patterns based on types of the merge conflicts detected by the semistructured merge tool that might occur when merging conflicting changes from two developers. Their study had similar purpose with the work in this thesis to understand the changes that lead to the merge conflicts. However, Accioly et al. [30] used a tool to analyze the changes and focused on general structural changes, while in this thesis we focus on all changes that led to the merge conflicts including inside method bodies. Furthermore, de Menezes [32] investigated the structure of the conflicting changes by analyzing language constructs such as method declaration and if statement. He conducted manual analysis on 5 Java projects and found most merge conflicts involve 4 language constructs. Also, de Menezes [32] found that there are 7 language constructs which occur frequently in conflicting changes in the merge conflicts. His work focused on the language constructs to categorize the merge conflicts.

Another study of merge conflicts focused on software practitioners' perspective on approach and resolution of the merge conflicts. McKee et al. [8] conducted interviews and a survey to understand factors that influence difficulty when resolving merge conflicts and how software practitioners approach merge conflicts. Their results show that complexity of the merge conflicts and experience in the project determine how software practitioners approach the merge conflicts. The software practitioners asses the merge conflicts and decide if they can resolve or revert the merge and merge later or pass to their colleagues. Moreover, their results show that lack of knowledge on the domain of conflicting code influence difficulty during merge conflicts resolution. This result is similar with our finding in the prestudy where developers reported that they had difficulty understanding conflicting changes during the resolution of the merge conflicts.

Studies of merge conflicts awareness focused on development of tools and strategies to avoid and reduce the occurrence of the merge conflicts. Guimarães and Silva [33] presented a tool WeCode which detects conflicts by continuously merging changes in a background system and notify developers without much disruption. The background system combines all changes (commited and uncommited) on developers' clone of a

project repository and from a master repository, while the developers are making changes. Similarly, Brun et al. [5] presented a tool called Crystal to help developers in conflict identification, management, and prevention. Their tool merges changes done (or committed) by developers on their working clones to a master repository in the background and reports status of the working copy. The developers can then decide to take right actions. Furthermore, Kasi and Sarma [7] presented a tool to proactive reduce conflicts by restricting concurrent changes on shared common files through task (planned change) orders. Their tool determines the tasks that would result to the merge conflicts and restricts developers from performing them so that to minimize the occurrence of merge conflicts.

Another study of merge conflict awareness by Sarma et al. [6] focused on awareness of changes in workspace artifacts, that is, files in a software configuration management system. They presented a tool that extends software configuration management systems to monitor changes on the artifacts. The information on the artifact changes are then shared to developers so that are aware of the modification which are currently done by others in the same artifact and, therefore, minimize conflicting changes. The implication of the results in this thesis focuses on designing a tool which provide information on conflicting changes to aid developers with the merge conflicts resolution.

3
Prestudy

This chapter reports on method, results, and discussion of the prestudy. The purpose of the prestudy was to explore challenges encountered by developers and how they resolved those challenges when integrating features into mainline branch from feature branches. The outcome of the prestudy led to further investigation of merge conflicts in the main study in order to understand what are changes led to their occurrence. In the prestudy, we observed that merge conflicts were the most reported challenge. We then further investigated the merge conflicts by replaying the integration of the features into the mainline branch so that to gain a deeper understanding. Also, related to the challenge of the merge conflicts, the developers reported that understanding implementation of the features in particular when resolving the merge conflicts was a challenge.

3.1 Method

An exploratory case study research method was adopted in the prestudy. The exploratory case study helps to find out what is happening, seeking new insights and generate ideas and hypothesis for new research [34]. The case study method is appropriate since the prestudy had an exploratory purpose as it aimed to explore challenges of integrating feature branches to mainline development project. Moreover, the case study method provides an approach which there is no precise boundary between phenomenon and study setting in comparison with experimentation [34].

In the prestudy, we explored Elasticsearch[1] project as a case, which is an open source full-text search and analytic engine that allows to store, search, and analyze big volumes of data quickly and in near real time [35]. The Elasticsearch project was chosen since it was used in a software evolution project in a masters program course and data were available. During the course from September to December 2016, 5 groups of 7 students integrated features from forks into a fork which was used as a mainline project.

A total of 16 features from 14 forks were integrated successfully during the course. Table 3.1 shows features and their forks used in the course. Table 3.2 shows size of the features before and after integration of the features. Also, it shows fork and mainline versions used by the groups during the feature integration.

[1] https://github.com/elastic/elasticsearch

Table 3.1: Features and their forks studied in prestudy

Feature	Fork
CancelableSearchAction	imotov/elasticsearch
CustomizableShardWeights	ywelsch/elasticsearch
FunctionScoreQueryFunctions	brwe/elasticsearch
GeoHeatMapAggregator	o19s/elasticsearch
GeoHeatMapGridAggregator	sstults/elasticsearch
GeoMeansAggregator	colings86/elasticsearch
SearchIngestProcessor	dadoonet/elasticsearch
NodesUsageAPI-REST	colings86/elasticsearch
ScrollPersistence	javanna/elasticsearch
StandardNumberAnalysis (Plugin)	jprante/elasticsearch-analysis-standardnumber
StringFieldsAnalyzerProcessor	imotov/elasticsearch
SynonymsGraph	mattweber/elasticsearch
TermsWithoutGlobalOrdinals	jpountz/elasticsearch
TopHitsAggregationSorting	martijnvg/elasticsearch
UnifiedHighlighter	jimczi/elasticsearch
xmlFormat	jprante/elasticsearch

Table 3.2: Features integrated by each group

Group	Feature	Feature size (LOC)		Version	
		Before Integration	After Integration	Fork	Mainline
A	GeoMeansAggregator	1011	1012	5.0.0-alpha6	5.0.0-alpha6
	NodesUsageAPI-REST	1592	1592	5.0.0-alpha6	5.0.0-alpha6
	xmlFormat	1138	1206	1.0.0.Beta1	5.0.0-alpha6
	StandardNumberAnalysis	10906	10257	2.3.3	5.0.0-alpha6
B	TopHitsAggregationSorting	42	42	5.0.0-alpha4	5.0.0-rc1
	GeoHeatMapAggregator	1204	1201	6.0.0-alpha1	5.0.0-rc1
C	GeoHeatMapAggregator	1204	1201	6.0.0-alpha1	5.0.0-alpha6
	SynonymsGraph	3045	3081	6.0.0-alpha1	5.0.0-alpha6
	UnifiedHighlighter	1145	1435	6.0.0-alpha1	6.0.0-alpha1
D	FunctionScoreQueryFunctions	501	625	5.0.0-alpha3	6.0.0-alpha1
	CustomizableShardWeights	186	183	5.0.0-alpha4	6.0.0-alpha1
	GeoHeatMapGridAggregator	1363	1363	6.0.0-alpha1	6.0.0-alpha1
	NodesUsageAPI-REST	1592	1592	5.0.0-alpha6	5.0.0-alpha6
	TermsWithoutGlobalOrdinals	314	315	2.0.0	6.0.0-alpha1
	TopHitsAggregationSorting	42	46	5.0.0-alpha4	6.0.0-alpha1
E	CancelableSearchAction	959	951	6.0.0-alpha1	5.0.0-alpha6
	StringFieldsAnalyzerProcessor	960	952	5.0.0-alpha6	5.0.0-alpha6
	ScrollPersistence	614	872	5.0.0-alpha6	5.0.0-alpha6
	SearchIngestProcessor	837	963	5.0.0-alpha6	5.0.0-alpha6

3.1.1 Data Collection

In the prestudy, both qualitative and quantitative data were collected. A mixture of qualitative and quantitative data provides more useful result compared to using only one of them [36]. The prestudy used two data collection methods. The methods are archival data and semi-structured interviews.

Archival Data

Archival data used in the prestudy included source code repositories and documentation. The source code of the groups projects were hosted in Github[2] private repositories. Access to the groups repositories was granted by a teacher responsible for the course. For each feature, the commits that integrated the feature into the mainline project were identified manually using information obtained from the reports, branches used to integrate the features and through repository's commit messages. After the feature commits were identified, size of the features before and after integration were extracted. To extract the feature sizes before and after integration, Git command `git-cherry-pick`[3], which applies changes from one commit or branch into another commit or branch, was used to apply the feature commits into new feature branches.

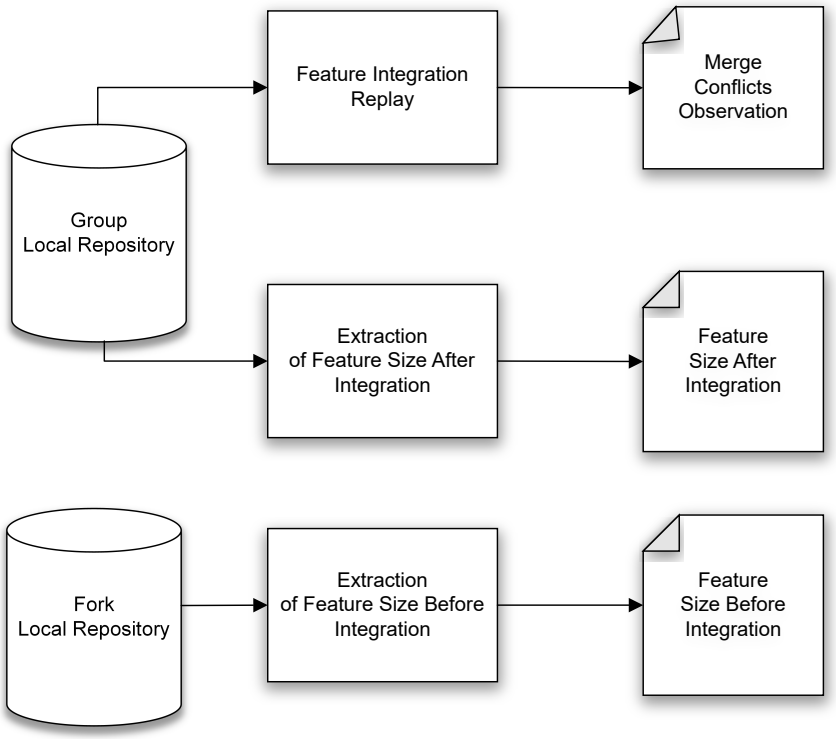

Figure 3.1: Extraction of data from source code repositories

Figure 3.1 shows overview of data extraction from source code repositories. In the extraction of the feature size, from any branch that contain feature code, a new

[2]https://github.com/
[3]https://git-scm.com/docs/git-cherry-pick

branch was created using a commit before the feature was integrated. This branch was used as a base before the integration. Another branch was then created that contains feature commits. Afterwards, branches before and after feature integration were compared using `git-diff`[4] command, which is used to show changes between two commits or branches. In order to easily compare the branches and obtain metrics such as Lines of Code (LOC) added for each features (see Table 3.3), we wrote a script and excluded documentation files.

Another data extraction process is replaying feature integration to find out what conflicts the groups had encountered. To achieve this, first, the merge commits hash were found and then they were used to determine the merge parents commits using the following Git command.

```
git log --pretty=%P -n 1 <merge commit hash>
```

Then, after merge parents hash were found, the following Git commands were used to replay the integration scenarios.

```
1 git checkout <parent1 hash>
2 git merge --no-commit --no-ff <parent2 hash>
3 git mergetool # if there is a conflict and need to visualize it
4 git merge --abort
5 git checkout <any branch>
```

The First step was to *checkout* or switching to the first merge commit parent, then followed with merging of the second commit parent without committing changes. Then, Git will determine whether there is a conflict or not by issuing a status message. If there is a conflict then any merge tool can be launched for conflict analysis and resolution. The next step was to *abort* the merge without saving the changes as the aim was only analysis of the merge without affecting the repository history. Then a last step was to *checkout* to any branch after replaying the merge. The source code provided both quantitative data on the size of the features before and after integration and qualitative data on the merge conflicts encountered while integrating the features.

Another source of the archival data is documentation. Documentation data in the prestudy referred to groups reports. The reports were written as part of the assessment of the course and provided qualitative data about the features and their forks and what challenges were encountered while integrating the features. Moreover, the reports provided qualitative data on how they successful integrate the features despite of the challenges.

Interviews

Interviews are important source of data in a case study since nearly all case studies involve human subjects [37]. In the prestudy, semi-structured interviews were conducted. The semi-structured interviews allow to plan questions but not asking in the

[4]https://git-scm.com/docs/git-diff

Table 3.3: Feature Integration metrics

Group	Feature	LOC		File		
		Added	Deleted	Added	Deleted	Modified
A	GeoMeansAggregator	1012	0	7	0	1
	NodesUsageAPI-REST	1592	14	18	0	13
	xmlFormat	1206	0	6	0	5
	StandardNumberAnalysis	10257	1	37	0	2
B	TopHitsAggregationSorting	42	37	0	0	3
	GeoHeatMapAggregator	1201	0	9	0	2
C	GeoHeatMapAggregator	1201	0	9	0	2
	SynonymsGraph	3081	1	8	0	3
	UnifiedHighlighter	1435	435	6	0	12
D	FunctionScoreQueryFunctions	625	173	3	1	10
	CustomizableShardWeights	184	50	0	0	4
	GeoHeatMapGridAggregator	1363	0	9	0	1
	NodesUsageAPI-REST	1592	14	18	0	13
	TermsWithoutGlobalOrdinals	315	0	1	0	1
	TopHitsAggregationSorting	46	39	0	0	5
E	CancelableSearchAction	951	78	6	0	35
	StringFieldsAnalyzerProcessor	952	235	5	4	13
	ScrollPersistence	872	108	8	1	19
	SearchIngestProcessor	963	0	8	0	1

order they were written and the questions depend on development of the conversation [34]. The list of the planned questions is used to make sure all the questions are asked. The semi-structured interviews were appropriate since we aimed to explore additional information based on the response of the questions while at the same time having planned questions necessary to answer the research questions guided the prestudy.

Given that the prestudy started after the course had finished, we aimed at having an interview with at least a representative of each group. We selected 5 representatives for 4 groups excluding one group which the author of this thesis was part of it based on accessibility. Then, requests for scheduling an interview were sent. Further invitations were sent to other members of the groups after there was only one response. However, due to different circumstances, only 2 interviews were conducted. The interviews were conducted with the representatives of group B and D. One interview was face-to-face and another through a video call. The interviews took 30 to 45 minutes and were recorded with permission. The following questions were planned for the interviews.

- Did you integrate feature(s) during the project?
- How did you integrate features during the project?
- What challenges or complexities did you encounter while integrating the features?
- How did you resolve those challenges?
- How long did it take to successful integrate the features?

3.1.2 Data Analysis

The interviews were analyzed together with the groups reports. The interview recordings were reviewed for additional information, that is, clarification of information provided in the reports. Then, after the review of the interview recordings, a summary of the challenges and how the groups resolved the challenges was written. The summary included analysis of the groups reports.

The groups reports were analyzed for the most reported challenges and how groups resolve those challenges for each integrated feature. The analysis of the challenges encountered by groups were then combined with the interviews summary.

3.2 Result - Feature Integration Challenges

In the following, challenges encountered by groups during integration of the features are presented.

3.2.1 Understanding implementation of the features

Two groups reported that they had to understand how the features work in order for them to successful perform the integrations especially when there are merge conflicts. For example, during integration of CustomizableShardWeights feature, group D had to understand the implementation of this feature so that to know how to resolve

issues such as deprecated methods, added parameters to methods, and moved classes as they reported: "*Some understanding of the surroundings was needed to resolved these issues successfully. For example, commit 27a760f 'replaces the explicit boolean flag that is passed around everywhere to denote changes to the routing table'.*". To resolve this issue, the group reported that: *"the code had to be changed a bit to not accommodate the boolean flag and instead make use of the auto-tracking feature implemented".*

3.2.2 Missing core classes which features depend on

Another challenge reported is missing of some of the core classes which the features depend on due to some changes done such as refactoring of an API. This was reported by group A. For example during integration of xmlFormat, group A had to look at lower version before major architectural changes were made as they reported: *"... we had to go through the source code of older versions of Elasticsearch (up till version 2.1 while current is in version 5.x) in order to understand the logic behind the functionality of the feature. What we discovered was that since version 2.x Elasticsearch changed several of it components resulting in missing classes.".*
In order to successful integrate the feature, the group had to add missing methods which the integrated feature depend on as they reported that: *"we decided to carefully implement some of the missing methods in the classes we added, wherever this was possible. In order to integrate this feature we had to add a total of 6 classes."*

3.2.3 Merge conflicts

This challenge was reported by three groups. Group D had encountered merge conflicts in integration of four features. For example in integration of CustomizableShardWeights feature they reported that: *"Upon merging, there were about 10 conflicts that could not be automatically resolved"* and further stated during interview with a group member: *"We had merge conflicts. Also, we encountered errors when we tried to build after integrating one feature.".*
In most of the merge conflicts, the groups reported that they resolve with minor effort. However, a few merge conflicts required more work. For example in integration of NodesUsageAPI-REST feature, group D reported that: *"This feature ... was submitted as pull request ... However, it was not completely accepted because there were merge conflicts in the files Node.java and RestIndicesActionTests.java.... Indeed we came across merge conflicts in the files mentioned above, which were fixed with some effort."*

3.3 Discussion and Conclusion

The challenge of merge conflicts was reported by most groups. However, the groups reported that they did not had to put much effort on resolving the merge conflicts. To learn more about merge conflicts challenge, we further analyzed the integration scenarios to understand what was the conflicting part in the merge conflicts and what changes led to their occurrence. Furthermore, we analyzed the integration

scenarios to understand how they resolved the merge conflicts. In the following, an example of an analysis of the changes that led to the merge conflict is presented.

3.3.1 Example of the Merge Conflict Analysis

The analysis was done in a merge conflict encountered during replaying of the CustomizableShardWeights feature integrated by group D. The merge conflict occurred as a result of 2 changes made in mainline version and 2 changes made in fork version. In the mainline, a method named `relocate` changed to `relocateShard`. The change was made to reduced `RoutingNodes` public interfaces methods which update routing entries. The methods which manipulate routing nodes were reduced to four which are `initializeShard`, `startShard`, `relocateShard` and `failShard`. After a shard had been started, the `relocateShard` method starts its relocation to another node. Also, it initialize a target shard and assigning it.

Another change in the mainline version is added `allocation.changes()` parameter in `getShardSize` method. The change was added in order to include shard routing changes along with allocation changes so that to efficiently update allocation ids by looking to changed shards. In the fork version, a `shardWeight` variable was assigned to a number of remaining shards after shard removal from a source node. Then, the `shardWeight` was passed to `addShard` method so that to increase total shard weight when a shard is added to a current node. The added parameter does not affect other files as the `addShard` method is defined in static class `ModelNode` in the same file which has the conflict.

The Listing below show changes made in the mainline version and the fork version compared to a common ancestor.

Mainline:

```
// core/src/main/java/org/elasticsearch/cluster/routing/allocation/allocator/
    ↪ BalancedShardsAllocator.java

sourceNode.removeShard(shardRouting);
Tuple<ShardRouting, ShardRouting> relocatingShards = routingNodes.relocateShard(
    ↪ shardRouting, target.nodeId(), allocation.clusterInfo().getShardSize(
    ↪ shardRouting, ShardRouting.UNAVAILABLE_EXPECTED_SHARD_SIZE)
    ↪ , allocation.changes());

currentNode.addShard(relocatingShards.v2());
```

Common Ancestor:

```
sourceNode.removeShard(shardRouting);
Tuple<ShardRouting, ShardRouting> relocatingShards = routingNodes.relocate(
    ↪ shardRouting, target.nodeId(), allocation.clusterInfo().getShardSize(
    ↪ shardRouting, ShardRouting.UNAVAILABLE_EXPECTED_SHARD_SIZE));

currentNode.addShard(relocatingShards.v2());
```

Fork:

```
int shardWeight = sourceNode.removeShard(shardRouting);
Tuple<ShardRouting, ShardRouting> relocatingShards = routingNodes.relocate(
    ↪ shardRouting, target.nodeId(), allocation.clusterInfo().getShardSize(
    ↪ shardRouting, ShardRouting.UNAVAILABLE_EXPECTED_SHARD_SIZE));

currentNode.addShard(relocatingShards.v2(), shardWeight);
```

Resolution of the Merge Conflict:

To resolve this conflict, the group included both changes from mainline and fork. They decided to include both versions since the mainline version is latest compared to the fork version. It would require them to make a lot of changes which are not part of the integrated feature if they would use only fork version. The code below show the changes made as part of the resolution.

```
int shardWeight = sourceNode.removeShard(shardRouting);
Tuple<ShardRouting, ShardRouting> relocatingShards = routingNodes.relocateShard(
    ↪ shardRouting, target.nodeId(), allocation.clusterInfo().getShardSize(
    ↪ shardRouting, ShardRouting.UNAVAILABLE_EXPECTED_SHARD_SIZE)
    ↪ , allocation.changes());

currentNode.addShard(relocatingShards.v2(), shardWeight);
```

3.3.2 Conclusion

In the prestudy, merge conflicts was the most reported challenge during feature integration. Furthermore, another challenge was understanding the implementation of the features during merge conflicts resolution. Therefore, there is a need to explore further the merge conflicts by analyzing the changes which were made prior to the merge conflict. The analysis of the changes should help in the future to build a tool which assist developers when incorporating changes from the mainline project and integrators of the changes into the mainline project. The main study will adopt similar approach used to collect and analyze merge conflicts in the prestudy.

4

Methodology

This chapter describes an approach used to extract and analyze the merge conflicts and changes that led to their occurrence. The chapter starts with a description of an extraction process of merge commits which had conflict and then followed with an extraction and analysis of conflicting versions in the merge conflicts.

Figure 4.1 depicts a process followed in collecting and analyzing the merge conflicts. The process starts with obtaining a local clone of a project under study, followed with the extraction of merge commits with conflicts. Then conflicting versions are extracted from the merge commits with conflicts, followed with analysis of the conflicting versions. Lastly, merge conflicts categories and changes that led to the merge conflicts are reported.

To study the merge conflicts in an open source project, we chose ElasticSearch[1] project. It is among most popular Java projects in terms of number of people starred in Github[2]. The Elasticsearch project was cloned locally on 5th of September 2017 from a master branch which targeted version 7.0.0-alpha1 and has 5,455 Java files with total of 682,240 lines of code.

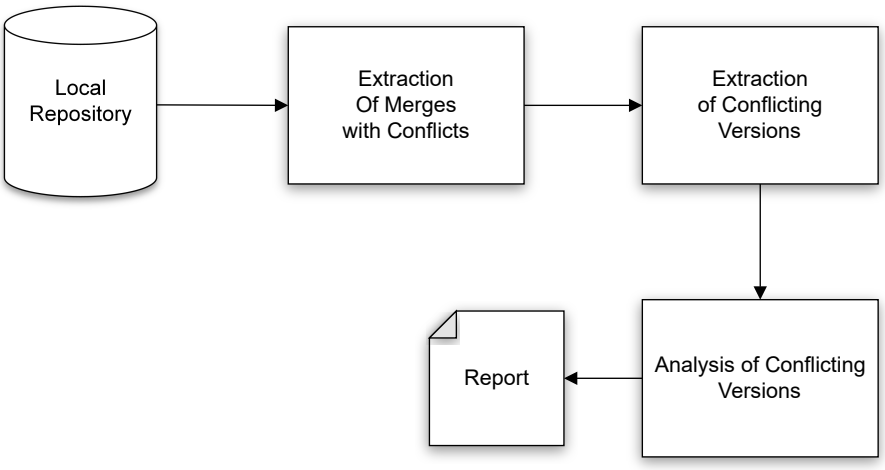

Figure 4.1: Overview of the Methodology

[1]https://github.com/elastic/elasticsearch
[2]https://github.com/

4.1 Extraction of Merges with Conflicts

To understand the merge conflicts and changes that led to their occurrence, we identified merge commits that had conflicting changes. However, we first identified merge commits in the project repository. A merge commit is the commit pointing to more than two commits as pointed out in Section 2.2.2. Therefore, the merge commits can be easily identified from other commits in a Git project.

Figure 4.2 shows an example of a Git commits history. In the example, the merge commit *C7* points to commits *C4* and *C6*. Git uses *three-way* merge, which is a merge of two branches, that is, a mainline and topic branch, using a recent common ancestor version if the mainline branch has changes after the topic branch has been created, otherwise it uses *fast forward* [26] which update a last commit in the mainline branch to point to a last commit of the topic branch.

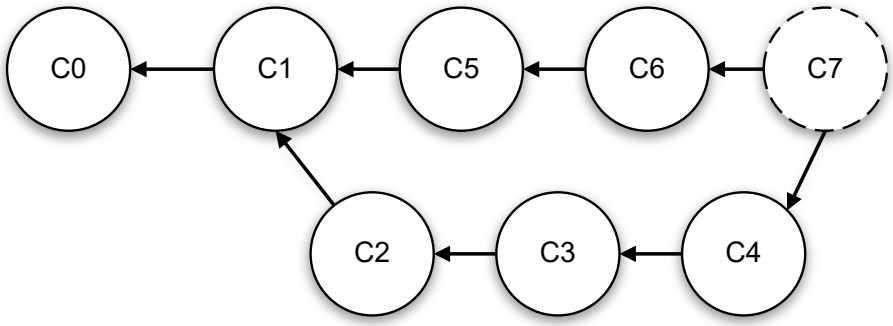

Figure 4.2: An example of a Git commits history. C0 stand for initial commit. C1 stand for common ancestor commit. C5 and C6 stand for mainline branch commits which are added after topic branch had been created. C2, C3, and C4 stand for topic branch commits. C7 stand for merge commit which points to tips (or last commits) of the mainline and topic branches.

In identifying merge commits with conflicting changes, we wrote a script which finds all merge commits in the project repository. We extracted the merge commits from both mainline and topic branches to capture merge conflicts which developers encountered while incorporating changes from the mainline branch. Also, in our initial analysis of feature integrations in the Elasticsearch repository, we observed that developers (contributors and members) mostly use *git rebase* which reapplies changes added in the topic branch into the mainline branch. Therefore, we included topic branches to have a larger sample of merge commits. Also, the *git rebase* command does not create a commit which points to two or more commits. Moreover, with *rebase*, when the changes are reapplied onto another branch, the merge conflicts encountered are resolved or a commit with the conflict is skipped, and resulted to a change in repository history. Therefore, replaying merges that used *rebase* to observe the merge conflicts occurred would be impossible or complex process.

While extracting the merge commits, we replayed merge scenarios using parent commits of the merge commits (see Git commands used to replay merge scenarios in Section 3.1.1). In the thesis work, we retrieved only merge commits with two parents, that is, a merge of two branches. However, though, in the project under study all the merges had only two parents. We used an option `--no-commit` with `git merge` command which tells Git not to create a merge commit. With this feature, which we used to discard merge commits without conflicts, Git allows to test a merge scenario without creating the merge commit. Therefore, we were able to test for each merge commit if there is a conflict without making changes in the project history.

4.2 Extraction of Conflicting Versions in Merge Conflict

In this phase, we used the dataset of the merge commits with conflicts retrieved from a previous phase. In the previous phase, we retrieved 260 merge commits with conflicts out of 2965 merge commits extracted from the Elasticsearch project. For each merge, we retrieved a merge commit hash, two parents commit hashes, and a merge date. Before extracting conflicting versions in the merge conflicts, we first conducted an initial analysis of the conflicting versions in a merge conflict. The aim of the initial analysis was to understand what is a conflicting part in the merge conflict and how the conflicting versions can be extracted using a script. The initial analysis was done using a sample of first 20 merge commits with conflicts dated from November 2016 to May 2017.

To visualize the merge conflicts in the initial analysis, we replayed the sample merge scenarios of the merge commits with conflicts. We improved the script used in the previous phase to automate the replaying of the sample merge scenarios. However, automation of previous and this phase can use one script, but we split into two scripts to simplify the process of extraction. We then configured a merge tool, which for each merge scenario was launched using a Git command `git mergetool`. We used Meld[3] merge tool which provides a *three-way* visualization when merging two versions of a file. With the *three-way* visualization, which shows a comparison of three versions, that is, the common ancestor, the mainline version, and the topic branch version, we were able to visualize the conflicting versions in the merge conflict. The Meld tool as a Git merge tool uses LOCAL, BASE, REMOTE, and MERGED versions to refer to a file with local changes, a file which is common to both branches, a file with remote changes, and a file which shows merge results respectively. Therefore, we used mainline branch, common ancestor, and topic branch versions to refer to LOCAL, BASE, and REMOTE changes respectively during a merge.

Figure 4.3 shows visualization of the merge conflict with three columns for the LOCAL, MERGED and REMOTE file respectively. Moreover, Figure 4.4 shows another tab configured to show LOCAL, BASE and REMOTE files so that to compare a difference between the conflicting versions.

[3]http://meldmerge.org/

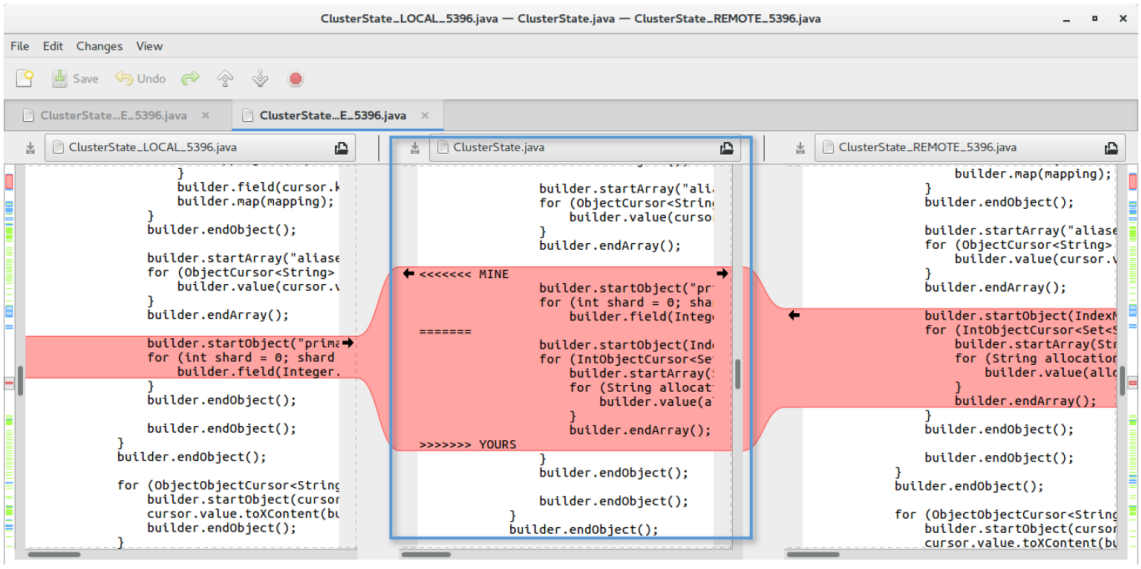

Figure 4.3: Merge conflict visualization using Meld. The blue frame shows MERGED version

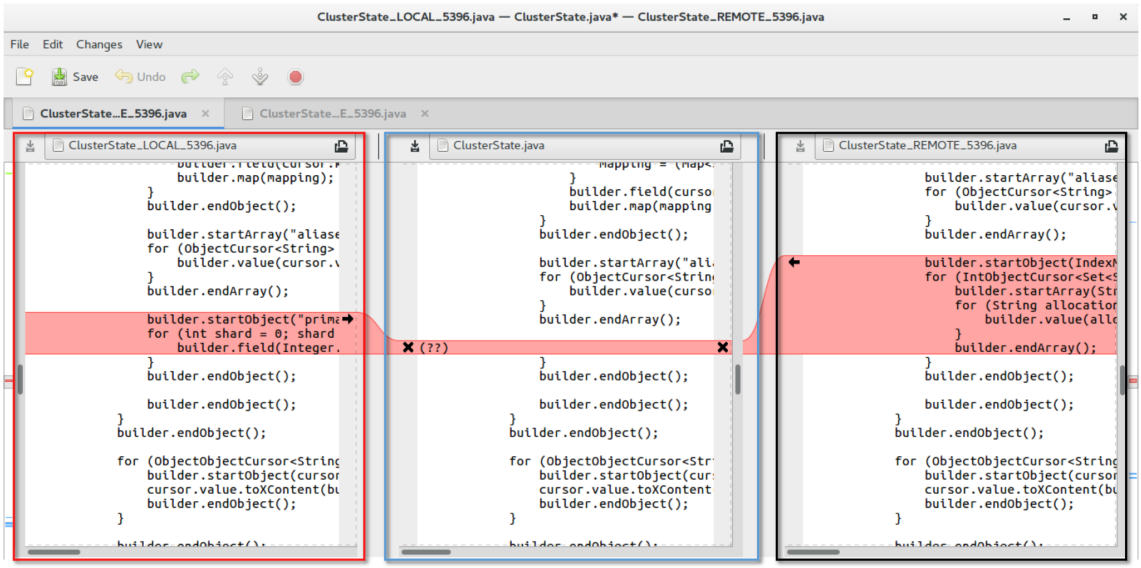

Figure 4.4: Conflicting changes visualization using Meld. The red frame shows LOCAL version, blue frame shows BASE version, and black frame shows REMOTE version

After initial analysis of conflicting versions, we extracted conflicting versions in all merge conflicts. However, in the initial analysis, we observed that there were merge conflicts due to whitespace and comment changes. Furthermore, there were merge conflicts due to ordering of elements such as methods in a source code file which is referred to as *ordering conflicts* [31]. Therefore, in order to minimize uninteresting

merge conflicts observed in the initial analysis, we configured a jFSTMerge[4] tool with Git. The tool adapts *semistructured merge* [31] which is a merge process that combine advantages of unstructured and structured merges when resolving merge conflicts. The unstructured merge does not take into account a structure of a program, which is an advantage, and therefore, leads to a problem of ordering conflicts. On the other hand, the structured merge take into account a programming language of a program, which provides more information that help to resolve most of the merge conflicts such as the ordering conflicts automatically which would otherwise required manual resolution. Therefore, with the configuration of jFSTMerge, we were able to eliminate most merge conflicts due to comment changes and ordering conflicts. However, the tool merges only Java files. Nevertheless, we were interested only with merge conflicts in Java files.

To retrieve conflicting versions, we adopted the script used in the initial analysis of conflicting versions. With the script, we collected 534 conflicting versions from the dataset of 260 merge commits with conflicts dated from beginning of June in 2015 to middle of April in 2017 in `core_module` of the Elasticsearch project. The script takes one conflict if there are similar conflicts (conflicts with same change) in a same file.

For the next phase, the following data were collected for each conflicting version: merge commit hash, merge parent commits hashes, merge date, conflict file, mainline branch version, and topic branch version.

4.3 Analysis of the Conflicting Versions

After extraction of conflicting versions in the merge conflicts, we analyzed the changes that led to the conflicts and categorized the merge conflicts. To perform an in-depth analysis of the changes and categorize the merge conflicts, we selected a sample of the conflict versions. Then, we analyzed the changes in project and code level perspectives, and categorized the merge conflicts based on the syntactic structure of the conflicting versions. The following sections describe the sample selection, merge conflict categorization and analysis of the changes that led to the merge conflicts.

4.3.1 Selection of Sample Conflicting Versions

To identify the sample for the in-depth analysis of the changes that led to merge conflicts and the merge conflict categorization, we randomly selected 40 conflicting versions from the dataset of the conflicting versions obtained in the previous phase. The following are procedures followed to determine the sample:

1. Assign unique numbers to 534 conflicting versions obtained in the previous step.

[4]https://goo.gl/RKe1jo

2. Use sample method with sample size 40 in LibreOffice Calc to randomly select the conflicting versions.
3. Identify the selected conflicting versions for the analysis of changes led to the merge conflict.

4.3.2 Categorizing Merge Conflicts

To categorize the merge conflicts, we first compared the conflicting versions, that is, the mainline and topic branch versions, with the common ancestor version. We then identified structural changes of the conflicting lines. Based on the observed structural changes, we derived initial merge conflict categories. For example, if in the mainline version a method is renamed and in the topic branch a parameter is added in the same method, then a category would be method rename and parameter addition. Therefore, it is a two-tuple (change in the mainline branch version, change in the topic branch version) or (change in the topic branch version, change in the mainline branch version). After we determined the initial categories, we then combined similar categories and refined the categories to remove duplicate. For example, method rename and parameter addition, and method parameter addition and method rename were combined into method rename and parameter addition. Furthermore, similar categories were grouped into shared structural characteristics such as method declaration.

4.3.3 Analysis of Project Level Changes that Led to the Merge Conflicts

To obtain the changes which were made in the conflicting lines, we used `git diff` command which shows changes between two branches. For example, from Figure 4.2, to obtain changes in a file which had a merge conflict in the mainline branch, a command `git diff C1:<conflict_file> C6:<conflict_file>` is used. On the other hand, the command `git diff C1:< conflict_file> C4:<conflict_file>` is used to obtain changes in the topic branch. The commit *C1* is the common ancestor of the two branches. After the changes in the conflict file were obtained, the changes that led to the merge conflicts were determined as the conflict file might have other changes not related to the merge conflict. These changes are referred to as *code level changes*.

After the changes in the source code were determined, we analyzed particular commit that made a change. To find the commit, here is referred to as *change commit*, we first searched the file which had the merge conflict in Github project repository using the merge parent commits. This means, for the changes in the topic and mainline branch we used a first merge parent and second merge parent respectively. This way we get the versions of the file before a merge. Then we used Github features *history* and *blame* to view the history and which commit made a change in the conflicting line. However, one can also use `git blame` command to view commits that made the change in a line in any file, but with Github it is easy to view and browse the change commit. Figure 4.5 shows an example of how the commits that made changes in the

conflicting lines was retrieved.

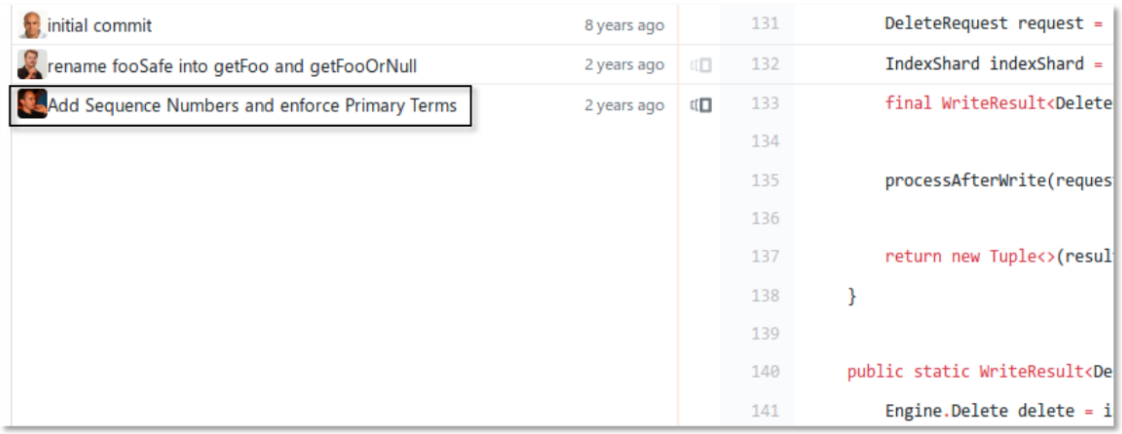

Figure 4.5: An example of retrieving a commit that made the change which led to the conflict. On the left side there are commits that modified the lines on the right side. By finding the line or lines that were involved in the merge conflict, a commit that made the change can be retrieved and a description of the change or changes will be analyzed.

After the change commit was retrieved, commit message and pull request associated with the change commit were analyzed for a change description. Then together with an investigation of the code level changes, we wrote the change description which are referred to as *project level perspective change*. Then a change category was derived based on a goal of the change. For example, if the goal of the change was to introduce a new feature, then the change will be categorized as feature introduction even if the refactoring task was done before the feature introduction. The commit and pull request messages helped to determine the change category by looking keywords such as refactor, improve, add, introduce, fix, remove, cleanup, problem, feature, separating, and test.

5

Results

This chapter presents the results of the analysis of conflicting versions in the merge conflicts. The chapter starts with the results of the merge conflict categorization and then followed by the results of the analysis of the changes that led to the merge conflicts. 40 sample merge conflicts were analyzed and changes that led to their occurrence were described. Appendix B.2 provides description of the changes that led to each merge conflict in the sample dataset. Table 5.1 shows an overview of the merge conflicts categories and changes that led to the merge conflicts identified for each merge conflict in the sample dataset.

5.1 Categories of Merge Conflicts

This section presents the results of the merge conflicts categorization. 6 categories were identified from the sample dataset. The merge conflicts were categorized based on characteristic of changes in both mainline and topic branch versions. For each identified category, a description and an example are provided. Table 5.2 shows a summary of the number of conflicts for each of the merge conflict category identified in the sample dataset.

5.1.1 Change of Method Call or Object Creation (MC_OC)

This category contains 5 subcategories. The subcategories are addition and/or removal of parameter values, addition or removal of parameter values and change of parameter value types, addition or removal and modification of parameter values, change of parameter values, and change of reference variable declaration. The subcategories are described below.

Addition and/or Removal of Parameter Values

There are three scenarios in this subcategory. First, a parameter value(s) is added in one version and removed in the other version. Second, the parameter value(s) is added in both versions. Third, the parameter value(s) is removed in both version. The change of the parameter value list is a result of changes in the parameter list where parameters are added or/and removed in the method or constructor declaration.

Example: In the mainline version, two parameter values are added, while in the topic branch version, one parameter value is added. This means two parameters were added in the mainline version and one parameter in the topic branch version to

the constructor parameter list.

Common ancestor version:

```
create = new Engine.Index(newUid("1"), doc, create.version(), create.versionType
    ↪ ().versionTypeForReplicationAndRecovery(), REPLICA, 0);
```

Mainline version:

```
create = new Engine.Index(newUid("1"), doc, create.version(), create.versionType
    ↪ ().versionTypeForReplicationAndRecovery(), REPLICA, 0, -1, false);
```

Topic branch version:

```
create = new Engine.Index(newUid("1"), doc, create.seqNo(), create.version(), create
    ↪ .versionType().versionTypeForReplicationAndRecovery(), REPLICA, 0);
```

Addition or Removal of Parameter Values and Change of Parameter Value Types

One or more parameter values are added or removed in one version, while in the other version, one or more parameter values are changed due to changes of parameter types.
Example: In the listings below, a parameter value changed from `index.version()` to `indexResult.getVersion()` due to parameter type change in the mainline version. In the topic branch version, a parameter value is added which implies a parameter was added in the parameter list.

Common ancestor version:

```
index = new Engine.Index(newUid("1"), doc, index.version(), index.versionType().
    ↪ versionTypeForReplicationAndRecovery(), REPLICA, 0, -1, false);
```

Mainline version:

```
index = new Engine.Index(newUid("1"), doc, indexResult.getVersion(), index.versionType
    ↪ ().versionTypeForReplicationAndRecovery(), REPLICA, 0, -1, false);
```

Topic branch version:

```
index = new Engine.Index(newUid("1"), doc, index.seqNo(), index.version(), index.
    ↪ versionType().versionTypeForReplicationAndRecovery(), REPLICA, 0, -1,
    ↪ false);
```

Addition or Removal and Modification of Parameter Values

A parameter value(s) is added or removed in one version and changed in the other version without changing the parameter type.

Example: In the mainline version, a parameter value `"test"` changed to `indexMetaData.getIndex()`, which returns an index name, and a boolean parameter value changed from `false` to `true`. In the topic branch version, a parameter value is added and a boolean parameter value changed from `false` to `true`.

Common ancestor version:

```
ShardRouting test_3 = ShardRouting.newUnassigned("test", 3, null, false, new
    ↪ UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "foo"));
```

Mainline version:

```
ShardRouting test_3 = ShardRouting.newUnassigned( indexMetaData.getIndex() , 3, null,
    ↪  true , new UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "foo"));
```

Topic branch version:

```
ShardRouting test_3 = ShardRouting.newUnassigned("test", 3, null, 1, true , new
    ↪ UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "foo"));
```

Change of Parameter Values

Change made on parameter values in both versions without changing parameter types. The change might be due to refactoring of the parameter values for readability purpose or other changes such as change of passed method call due to changes in the method declaration.
Example: In the mainline version, a parameter value of a method call was replaced with another method call which has different signature. In the topic branch version, two parameter values are improved to pass different kind of values.

Common ancestor version:

```
return new InternalMin(name, randomDouble(), randomFrom(DocValueFormat.BOOLEAN,
    ↪ DocValueFormat.GEOHASH, DocValueFormat.IP, DocValueFormat.RAW),
    ↪ pipelineAggregators, metaData);
```

Mainline version:

```
return new InternalMin(name, randomDouble(), randomNumericDocValueFormat() ,
    ↪ pipelineAggregators, metaData);
```

Topic branch version:

```
double value = frequently() ? randomDouble() : randomFrom(new Double[] { Double.
    ↪ NEGATIVE_INFINITY, Double.POSITIVE_INFINITY });
DocValueFormat formatter = randomFrom(new DocValueFormat.Decimal("###.##"),
    ↪ DocValueFormat.BOOLEAN, DocValueFormat.RAW);

return new InternalMin(name, value , formatter , pipelineAggregators, metaData);
```

Change of Reference Variable Declaration

Change of type and renaming of the reference variable. The reference variable may be renamed in one version and the type is changed in the other version, and vice versa. Also, in both versions, the variable type or name may be changed. The change of the reference variable may lead to change of method call.
Example: In the mainline version, a *generic* type is changed from `IndexShardInjectorPair` to `IndexShard`, while in the topic branch version, the variable name changed from `tmpShardsMap` to `newShards`.

Common ancestor version:

```
HashMap<Integer, IndexShardInjectorPair> tmpShardsMap = new HashMap<>(shards);
```

Mainline version:

```
HashMap<Integer, IndexShard> tmpShardsMap = new HashMap<>(shards);
```

Topic branch version:

```
HashMap<Integer, IndexShardInjectorPair> newShards = new HashMap<>(shards);
```

5.1.2 Change of an Assert Statement Expression (AS_EXP)

Change made on assert statement as a result of other changes such as refactoring of the class or method name, and changes on the strings which are passed in the assert expression.
Example: In the listings below, a parameter value passed in a method call which is the assert expression is changed in both the mainline and topic branch version.

Common ancestor version:

```
assertThat(e.getRootCause(), instanceOf(QueryParsingException.class));
```

Mainline version:

```
assertThat(e.getRootCause(), instanceOf(ParsingException.class));
```

Topic branch version:

```
assertThat(e.getRootCause(), instanceOf(QueryShardException.class));
```

5.1.3 Addition of Statements in the Same Area (ADD_STMT)

One or more statements are added in the same area of the code in both the mainline and topic branch version. The statements can be added in a control flow statement, method call or in object creation. The statements include: method calls, control flow

statements, object creation statements, assignment statements, assert statements, and increment statements.

Example: In the listings below, in both the mainline and topic branch version, a method call and `for` statement are added. In the mainline, the `for` statement has a nested `for` statement.

Common ancestor version:

```
for (IndexMetaData indexMetaData : metaData()) {
    builder.startObject(indexMetaData.getIndex(), XContentBuilder.
        ↪ FieldCaseConversion.NONE);
    ...
    builder.endArray();

    builder.endObject();
}
```

Mainline version:

```
for (IndexMetaData indexMetaData : metaData()) {
    builder.startObject(indexMetaData.getIndex(), XContentBuilder.
        ↪ FieldCaseConversion.NONE);
    ...
    builder.endArray();

    builder.startObject(IndexMetaData.KEY_ACTIVE_ALLOCATIONS);
    for (IntObjectCursor<Set<String» cursor : indexMetaData.getActiveAllocationIds()) {
        builder.startArray(String.valueOf(cursor.key));
        for (String allocationId : cursor.value) {
            builder.value(allocationId);
        }
        builder.endArray();
    }

    builder.endObject();
}
```

Topic branch version:

```
for (IndexMetaData indexMetaData : metaData()) {
    builder.startObject(indexMetaData.getIndex(), XContentBuilder.
        ↪ FieldCaseConversion.NONE);
    ...
    builder.endArray();

    builder.startObject("primary_terms");
    for (int shard = 0; shard < indexMetaData.getNumberOfShards(); shard++) {
        builder.field(Integer.toString(shard), indexMetaData.primaryTerm(shard));
    }
    builder.endObject();
}
```

5.1.4 Modification and Removal of Statements (MOD/RMV_STMT)

The statements are modified in one version and removed in the other version. The modification of the statements include addition and removal of statements inside control statements, that is, branching statements, decision-making statements, and looping statements. It also include exception handler blocks, that is, `try`, `catch`, and `finally` blocks. Therefore, modification in this category means any changes that are made in a statement while that statement is removed in the other version. The changes may be due to refactoring such as splitting the logic of the implementation inside method which leads to removal of statements, and additional of new logic due to a new added feature, which adds or modifies statements.
Example: In the listings below, an object creation statement is added inside the `try` block in the topic branch version, while in the mainline version, the `try` block is replaced with `if` statement which has `switch` statement.

Common ancestor version:

```
public <T extends Throwable> T readThrowable() throws IOException {
    try {
        ObjectInputStream oin = new ObjectInputStream(this);
        return (T) oin.readObject();
    } catch (ClassNotFoundException e) {
        throw new IOException("failed to deserialize exception", e);
    }
}
```

Mainline version:

```
public <T extends Throwable> T readThrowable() throws IOException {
    if (readBoolean()) {
        int key = readVInt();
        switch (key) {
            case 0:
            ...
            case 17:
            return (T) readStackTrace(new LockObtainFailedException(readOptionalString(),
                ↪ readThrowable()), this);
            default:
            assert false : "no such exception for id: " + key;
        }
    }
    return null;
}
```

Topic branch version:

```
public <T extends Throwable> T readThrowable() throws IOException {
    try {
        ObjectInputStream oin = new ObjectInputStream(this);
        @SuppressWarnings("unchecked")
```

```
            T object = (T) oin.readObject();
            return object;
        } catch (ClassNotFoundException e) {
            throw new IOException("failed to deserialize exception", e);
        }
    }
```

5.1.5 Changes in Different Statements in the Same Area (D_STMT)

Changes that are made in different statements which are located next to each other. **Example:** In the listings below, in the mainline version, a condition in the `if` statement is changed while in the topic branch, a parameter value of a method call is changed. The `if` statement and method call statement are next to each other.

Common ancestor version:

```
public void doXContent(XContentBuilder builder, Params params) throws IOException
    ↪ {
    builder.startObject(PrefixQueryParser.NAME);
    if (boost == -1 && rewrite == null && queryName != null) {
        builder.field(name, prefix);
    }
    ...
}
```

Mainline version:

```
public void doXContent(XContentBuilder builder, Params params) throws IOException
    ↪ {
    builder.startObject(PrefixQueryParser.NAME);
    if (boost == -1 && rewrite == null && queryName == null) {
        builder.field(name, prefix);
    }
    ...
}
```

Topic branch version:

```
public void doXContent(XContentBuilder builder, Params params) throws IOException
    ↪ {
    builder.startObject( NAME );
    if (boost == -1 && rewrite == null && queryName != null) {
        builder.field(name, prefix);
    }
    ...
}
```

5.1.6 Change of IF Statement condition (IF_C)

Changes made in the `if` statement condition(s) in both conflicting versions. The changes may be due to changes in the method call used as the condition which also may be changed due changes in the method declaration.
Example: In the mainline version, a condition is changed due to changes in the method call as a result of method renaming, while in the topic branch version, the

conditions are replaced with shorter condition as the result of the logic implementation refactoring.

Common ancestor version:

```
if (nodeSpecificClusterState.version() < currentState.version() && Objects.equal(
    ↪ nodeSpecificClusterState.nodes().masterNodeId(), currentState.nodes().
    ↪ masterNodeId())) {
    return currentState;
}
```

Mainline version:

```
if (nodeSpecificClusterState.version() < currentState.version() && Objects.equals(
    ↪ nodeSpecificClusterState.nodes().masterNodeId(), currentState.nodes().masterNodeId()) ) {
    return currentState;
}
```

Topic branch version:

```
if ( currentState.supersedes(nodeSpecificClusterState) ) {
    return currentState;
}
```

5.2 Changes that led to the Merge Conflict

This section presents the results of the analysis of project level changes that led to the merge conflicts. In the analysis, for each conflicting change in mainline and topic branch, a category was identified based on the description of the conflicting changes. From the understanding of the changes, 8 change categories were identified from the sample dataset. Table 5.3 shows a summary of the number of changes for each category in both the mainline and topic branch.

5.2.1 Feature Introduction

A change which introduces or is part of introducing a new feature. The change may include tests added for the feature and necessary refactoring.
Example: During introduction of a WriteOperationsSequenceNumbers feature, which track how many times a replica shard was promoted to a primary shard from existing replica shards after existing primary shard has failed, a counter was added to the write operations such as indexing. The counter is used for identification of the operations from the failed primary shard so that they cannot be executed. The listing below shows added method call for counting number of the write operations on the primary shard. The counter `delete.seqNo()` for the delete operation is passed to the sequence number of the primary shard. Also, the counter is passed with the response of the delete operation.

```
+    request.seqNo(delete.seqNo());
```

```
-   DeleteResponse response = new DeleteResponse(indexShard.shardId(), request.
    ↪ type(), request.id(), delete.version(), delete.found());
+   DeleteResponse response = new DeleteResponse(indexShard.shardId(), request.
    ↪ type(), request.id(), delete.seqNo(), delete.version(), delete.found());
```

5.2.2 Refactoring

A change which is not part of a particular feature introduction or enhancement, but as part of the maintenance. The change includes redesigning of an API, extraction of a method or class, replacing control flow statements, and renaming variable, class or method.

Example: A logic for creating a REST response such as xContent from a response of the write operation such as delete, which is shared by the write operations, was extracted to a class. Listing below shows changes as part of extracting shared logic. The logic passed when building the xContent response for the response of the `update` operation using xContent builder was replaced with an object of the class `RestStatusToXContentListener` which returns the xContent response.

```
-   client.update(updateRequest, new RestBuilderListener<UpdateResponse>(channel)
    ↪ {
-       @Override
-       public RestResponse buildResponse(UpdateResponse response, XContentBuilder
    ↪  builder) throws Exception {
-           builder.startObject();
-           ActionWriteResponse.ShardInfo shardInfo = response.getShardInfo();
-           builder.field(Fields._INDEX, response.getIndex())
-                   .field(Fields._TYPE, response.getType())
-                   .field(Fields._ID, response.getId())
-                   .field(Fields._VERSION, response.getVersion());
-
-           shardInfo.toXContent(builder, request);
-           if (response.getGetResult() != null) {
-               builder.startObject(Fields.GET);
-               response.getGetResult().toXContentEmbedded(builder, request);
-               builder.endObject();
-           }
-
-           builder.endObject();
-           RestStatus status = shardInfo.status();
-           if (response.isCreated()) {
-               status = CREATED;
-           }
-           return new BytesRestResponse(status, builder);
-       }
-   });
+   client.update(updateRequest, new RestStatusToXContentListener<>(channel));
```

5.2.3 Feature Enhancement

A change which is added or part of addition to the existing feature. The change may improve the quality of the feature or extend the ability of the feature.

Example: The ShardAllocationIDs feature, which generates IDs when the shards are allocated to a cluster and are used for the shards recovery when the cluster is restarted, was enhanced so that the allocation IDs are carried on to the Index metadata, which is the metadata of the Index retrieved from the cluster state information. The allocation IDs are then used during selection of a new primary

shard. Listing below shows changes added so that the allocation IDs can be carried on to index metadata in the cluster state.

```
+   builder.startObject(IndexMetaData.KEY_ACTIVE_ALLOCATIONS);
+   for (IntObjectCursor<Set<String>> cursor : indexMetaData.
    ↪ getActiveAllocationIds()) {
+       builder.startArray(String.valueOf(cursor.key));
+       for (String allocationId : cursor.value) {
+           builder.value(allocationId);
+       }
+       builder.endArray();
+   }
```

5.2.4 Test Improvement

A change that improves or corrects one or more tests. This do not add new tests as they are part of feature introduction or feature enhancement. The change may correct or add new test infrastructure.
Example: The aggregations tests were modified so that not to test metrics aggregations such as Min aggregation that expect a numeric value from the document field datatypes such as IP address which are not numeric. The improvement was made as the tests were returning wrong aggregations. Listing below shows the improvement on the creation of the Min aggregation test instance. The method call which returns random selected document field datatype was replaced with the method which returns random selected numeric datatype.

```
-   return new InternalMin(name, randomDouble(),
-           randomFrom(DocValueFormat.BOOLEAN, DocValueFormat.GEOHASH,
    ↪ DocValueFormat.IP, DocValueFormat.RAW), pipelineAggregators, metaData);
+   return new InternalMin(name, randomDouble(), randomNumericDocValueFormat(),
    ↪ pipelineAggregators, metaData);
```

5.2.5 Bug Fix

A change that corrects wrong requirement, implementation of the requirement, and logic such as sequencing of the statements.
Example: The logic implemented in a Java API builder resulted to missing field when the queries are serialized in Json format in Index queries. An Index query has fields such as `_name` which may be used to provide a name for the query. Listing below shows a change that was made in the builder which fix the logic for building a query field using the builder `name` if a query name is empty. Before the change, the query field is created if the query name was empty which resulted to other fields such as `prefix` to not be created.

```
-   if (boost == -1 && rewrite == null && queryName != null) {
+   if (boost == -1 && rewrite == null && queryName == null) {
        builder.field(name, prefix);
    } else {
        builder.startObject(name);
        builder.field("prefix", prefix);
        if (boost != -1) {
            builder.field("boost", boost);
        }
        if (rewrite != null) {
            builder.field("rewrite", rewrite);
        }
```

```
        if (queryName != null) {
            builder.field("_name", queryName);
        }
        builder.endObject();
    }
```

5.2.6 Framework Removal

A change that removes a framework or frameworks from the project code base. The change may include the refactoring of the API after the framework has been removed.
Example: A framework Guice[1] which is used for dependency injection in Java version 6 and above was removed from the Elasticsearch project code base. As part of the Guice framework removal, the Shard query and request cache modules were moved to Indices service which is used for managing and monitoring of Indices. Listing below shows a change which was part of removing the Guice framework. After the framework has been removed, a parameter value `indicesService.getIndicesQueryCache()` which pass the Indices query cache when creating a new object for the statistics that are common for all Shards in the Node.

```
-   shardsStats.add(new ShardStats(indexShard.routingEntry(), indexShard.shardPath
    ↪ (), new CommonStats(indexShard, SHARD_STATS_FLAGS), indexShard.commitStats
    ↪ ()));
+   shardsStats.add(new ShardStats(indexShard.routingEntry(), indexShard.shardPath
    ↪ (), new CommonStats(indicesService.getIndicesQueryCache(), indexShard,
    ↪ SHARD_STATS_FLAGS), indexShard.commitStats()));
```

5.2.7 Breaking Change Fix

A change that corrects the changes which break the usage of the API. The breaking change may affect backward compatibility and may require actions such upgrading the version of the software.
Example: A metric in the Cluster statistics which is used to show available memory that can be used by all Nodes in the Cluster was removed as it remained when a change to remove statistics that are specific for a certain Operating system was made. Listing below shows part of the change to remove the available memory metric to fix the breaking change that removed specific operating system statistics.

```
-   availableMemory = in.readLong();
```

5.2.8 Library Removal

A change that removes a library or libraries from the project code base. The change may be followed with required refactoring.
Example: A library Guava[2] which contains the Java core libraries was removed from the Elasticsearch code base. As part of the Guava removal, listing below shows part of the change to remove the Immutable map which is part of the Guava library collection. After the change, the Immutable map uses Java Collections class.

[1]https://github.com/google/guice
[2]https://github.com/google/guava

```
-    shards = ImmutableMap.copyOf(tmpShardsMap);
+    shards = unmodifiableMap(newShards);
```

Table 5.1: Overview of merge conflicts and categories of changes led to the merge conflicts in the dataset

Conflict #	Conflict Category	Change Category	
		Mainline	Branch
1	MC_OC	Test improvement	Feature introduction
2†	MC_OC, ADD_STMT	Refactoring	Feature introduction
3	ADD_STMT	Refactoring	Feature introduction
4	MC_OC	Framework removal	Feature introduction
5	MC_OC	Refactoring	Feature introduction
6	MC_OC	Refactoring	Feature introduction
7	MC_OC	Feature enhancement	Feature introduction
8	AS_EXP	Feature enhancement	Feature introduction
9	MOD/RMV_STMT	Bug fix	Feature introduction
10	ADD_STMT	Refactoring	Feature introduction
11	MOD/RMV_STMT	Refactoring	Refactoring
12	MC_OC	Refactoring	Feature introduction
13	MC_OC	Breaking change fix	Feature enhancement
14	MC_OC	Refactoring	Feature enhancement
15	MC_OC	Feature enhancement	Test improvement
16	MC_OC	Feature enhancement	Refactoring
17	ADD_STMT	Feature enhancement	Feature enhancement
18	MC_OC	Framework removal	Feature introduction
19	MC_OC	Feature enhancement	Feature introduction
20	MC_OC	Feature enhancement	Feature introduction
21	MC_OC	Feature enhancement	Feature introduction
22	MC_OC	Bug fix	Refactoring
23	MC_OC	Feature enhancement	Feature introduction
24	MC_OC	Test improvement	Feature enhancement
25	ADD_STMT	Bug fix	Bug fix
26	ADD_STMT	Refactoring	Refactoring
27	MC_OC	Refactoring	Feature introduction
28	MC_OC	Refactoring	Feature introduction
29	MC_OC	Refactoring	Feature introduction
30	ADD_STMT	Feature enhancement	Feature introduction
31	ADD_STMT	Feature enhancement	Feature introduction
32	MC_OC	Feature enhancement	Feature introduction
33	MC_OC	Refactoring	Library removal
34	AS_EXP	Refactoring	Refactoring
35	ADD_STMT	Refactoring	Refactoring
36	MC_OC	Refactoring	Refactoring
37	IF_C	Library removal	Refactoring
38	MC_OC	Refactoring	Test improvement
39	MOD/RMV_STMT	Refactoring	Refactoring
40	D_STMT	Bug fix	Refactoring

† conflict has two categories

Table 5.2: Merge conflict categories identified in the sample dataset

Conflict Category	# of Conflict
MC_OC	25
AS_EXP	2
ADD_STMT	9
MOD/RMV_STMT	3
D_STMT	1
IF_C	1

Table 5.3: Changes that led to the merge conflicts identified in the sample dataset

Change category	# of Changes	
	Mainline	Topic Branch
Feature Introduction	0	22
Refactoring	18	10
Feature Enhancement	12	4
Test Improvement	2	2
Bug Fix	4	1
Framework Removal	2	0
Breaking Change Fix	1	0
Library Removal	1	1

6
Discussion

This chapter discusses the results of the analysis of changes led to the merge conflicts and categorization of the merge conflicts. Furthermore, it discusses the threat to validity of the study. The chapter begins with the discussion of the categorization of the merge conflicts and followed with the changes led to the merge conflicts. Then it ends with the discussion of the threats to validity.

6.1 Categorizing Merge Conflicts

Most merge conflicts (25 out of 40) in the sample dataset were categorized as *method call or object creation* (MC_OC). The method call and object creation were grouped as one category since conflicting changes are related to parameter values and variables that are assigned to method calls or objects creation. Accioly et al. [30] found that most merge conflicts (85%) are due to changes that are made in method bodies. Their finding relates to our results in that all merge conflicts in the sample dataset occurred in the method bodies. However, in our results, we cannot generalize that most changes which led to merge conflicts were made inside method bodies. Nevertheless, our results may shed light on a suggestion by Accioly et al. [30] to investigate what changes led to the merge conflicts in the method bodies. Furthermore, de Menezes [32] found that *method invocation* is a most language construct involved in merge conflicts. His finding relates with our finding in that most conflicting changes are in MC_OC category which includes changes in method calls.

Whereas most merge conflicts are due to changes in method calls or objects creation, we would have expected to see merge conflicts in method or constructor signatures alongside the merge conflicts that occurred in their method call or object creation statements. The reason is that, for example, if developers made changes such as adding or removing parameter values it means they have added or removed parameters in a method signature. Therefore, we would like to research this in more projects to study if there is a merge conflict in a method call or object creation statement due to added or removed parameter values then there should also be a merge conflict in the method or constructor signatures and vice versa.

Our results of merge conflict categories relate to two merge conflict patterns found by Accioly et al. [30]. First, Accioly et al. [30] found a pattern of methods or constructors added with the same signature and different bodies. This pattern is

similar to MC_OC category since it involves changes inside methods. However, in our sample dataset there are no merge conflicts due to changes in method or constructor declaration. Second, they found a pattern of different edits to the same or consecutive lines of the same method or constructor. This pattern relates to MC_OC, AS_EXP, and D_STMT categories as they are based on changes performed on same or adjacent statements. Other patterns found by Accioly et al. [30] are different edits to the same class field declaration; class fields declarations added with the same identifier and different types or modifiers; different edits to the modifier list of the same type declaration (class, interface, annotation or enum types); different edits to the same implements declaration; different edits to the same extends declaration; different edits to the same enum constant declaration; and different edits to the same annotation method default value declaration. In our sample dataset there are no merge conflicts that involve changes to declaration of class, class fields, interface, enum, annotation, enum constant, and extends.

Furthermore, de Menezes [32] found most conflicting changes that led to merge conflicts involve the following language constructs: method invocation, method declaration, method signature, variable, import, if statement, and commentary. However, classification of merge conflicts by de Menezes [32] involve combination of all unique language constructs in conflicting changes. Using the language constructs, de Menezes [32] found the following most kind of conflicts: method invocation; import; method invocation, variable; method declaration; variable; if statement; method signature; and if statement, method invocation, variable. The method invocation and variable kind of conflicts may relate to MC_OC category. Also, if statement kind of conflict may relate to ADD_STMT, MOD/RMV_STMT, D_STMT, and IF_C categories. However, in our sample there are no merge conflicts involve import, method declaration, and method signature language constructs. Also, merge conflicts that involved only comments were ignored in this study.

Most merge conflicts were due to minor changes in terms of their size such as addition of parameter values. This may be due to developers in the project under study were working on updated copy, which means they often incorporate changes from a mainline branch into their individual branches. However, some merge conflicts were regarded as complex since they involved more than one line. These merge conflicts were categorized as *Addition of Statements in the Same Area* (ADD_STMT) and *Addition and Removal of Statements* (MOD/RMV_STMT), and they account for 12 out of 40 analyzed merge conflicts. Despite of the merge conflicts being regarded as minor in terms of the size, they may have led to more changes during their resolution. Therefore, all the merge conflicts were important in the analysis of changes that led to their occurrence.

Moreover, merge conflicts in D_STMT and ADD_STMT categories may be resolved automatically by structured merge tools such as JDime [38] since there are no conflicting changes on a same statement.

6.2 Changes that lead to the merge conflicts

Most changes that led to the merge conflicts in the mainline branch were due to *refactoring* (18 out of 40) and in the topic branch were due to *feature introduction* (22 out of 40). We found no change that led to the merge conflict in the mainline branch which is due to feature introduction. This may be because of the retrieved merge conflicts occurred in the feature branches and there were no conflicting changes added in the mainline branch as part of the feature introduction. Second most changes that led to the merge conflicts in the mainline branch were due to *feature enhancement* (12 out of 40) and in the topic branch were due to *refactoring* (10 out of 40). In total, the refactoring changes accounted for most changes led to merge conflicts. This may indicate that the refactoring task accounted for most of the merge conflicts.

As seen in Table 5.1, there are 6 pairs of refactoring changes. This means merge conflicts occurred due to changes made as part of refactoring in both the mainline and topic branches. Other pairs include feature enhancement and *bug fix* which are both one pair each. For the bug fix change, it is surprising that different bugs are fixed by changing same part or line of the code. Further investigation may be conducted to understand whether the bugs are related or not.

Most of the change categories are related or depend on one another. The changes that led to the merge conflict can be categorized into 3 change types which are feature introduction, bug fix, and refactoring similar to the categories used by [39]. However, we chose to use feature enhancement, library removal, framework removal, test improvement, and breaking change fix categories which are subtype of feature introduction, refactoring, and bug fix to understand merge conflicts which are due to the subtypes of the change types as they may be rare in the project. The feature enhancement category can be part of feature introduction and breaking change can be part of bug fix. Moreover, the test improvement, framework removal, and library removal categories can be part of refactoring. These changes may be part of the code base maintenance. Furthermore, the categorization of the changes was based on the goal of the change, for example, the goal of a change may be to introduce a new feature, but it may have required to refactor some methods by extracting common logic which the new feature shares with existing feature(s) and, therefore, the introduction of the feature would be proceeded with refactoring. Similarly, the *test improvement* change may be done after introduction of the feature or as part of the refactoring. Also, the refactoring task may be performed during bug fixing.

6.3 Threats to Validity

There is no study without any threat to its validity. This section discusses the following possible threat to validity of the study: *construct validity*, *internal validity* *external validity*, and *reliability* as presented by Runeson and Höst [34].

6.3.1 Construct Validity

This validity threat concerns the connection between research questions and what is observed by the researcher [34]. A possible threat to construct is the categorization of the merge conflicts based on the changes that led to the merge conflict. There is a risk the conflicts might be categorized different based on understanding of the changes. To address this threat, we first created a pair of changes, that is, in the mainline and topic branch. Then we identified similar characteristics based on the language syntax. However, still, there might be a risk that the merge conflicts are placed in wrong categories.

6.3.2 Internal Validity

This validity threat concerns the factors that the researcher may not be aware how much they affect the factors under investigation [34]. There is a risk of the scripts written to extract merges that had conflicts and conflicting versions in those merges and the jFSTMerge tool, which was configured with Git to reduce merge conflicts related to whitespace and comment changes, to contain defects or bugs. Therefore, it might lead to false merge conflicts which might affect the in-depth analysis of the conflicting changes. However, to reduce this threat, we tested the scripts several times to make sure the right merge conflicts are extracted and manually reviewed the merge conflicts before the analysis phase to remove the false positive merge conflicts.

Another possible threat to internal validity is collection of the changes in the project code base that led to the conflicts. There might be errors when analyzing the changes and led to wrong result. However, to reduce this threat, a thorough analysis was conducted using more than one source to understand the changes. The commit and pull request messages together with source code were used to obtain the understanding of the changes.

6.3.3 External Validity

This validity threat concerns the generalization of the findings to other cases [34]. An in-depth analysis of the changes that led to the merge conflicts and categorization of the merge conflicts were done using a sample of 40 merge conflicts. There is a threat that the selected sample of conflicting changes may not represent the conflicting changes extracted from the merge commits which had conflicts. Therefore, the results of the in-depth analysis might not be generalized to the subject system under study. However, to mitigate or reduce this threat, we randomly reviewed the conflicting changes that were not selected for the in-depth analysis to observe whether there are cases that were not represented especially for merge conflict categorization.

Furthermore, the analysis of the merge conflicts was based on one project. Therefore, we cannot generalize the results of the study to other projects. Also, since the studied project is Java project and the results of the merge conflicts categorization was based on Java syntax, therefore, the results of the merge conflicts categorization would be different in other programming languages. Moreover, the studied project was an

open source project, thus the results might not be generalized to a closed source project. Furthermore, there might be a bias when choosing a subject system for this study. Therefore, we might have had different results if we would have chosen a different project. However, we believed the Elasticsearch project was interesting project given its popularity in Github and its size.

6.3.4 Reliability

This threat concerns the replication of the study by other researchers to obtain same results [34]. To mitigate this threat, we described the methodology used in this study with replication in mind and made available the scripts used to extract the merge conflicts and the conflicting versions in the merge conflicts. Also, we made available the dataset of the merge which had conflicts and the conflicting versions so that the researchers can compare their results and ours. However, there is a risk that the steps we followed to extract and analyze the merge conflicts might not be clear to some researchers when replicating our study.

7

Conclusion and Future Work

This chapter presents the conclusion of the study and discusses the proposed future work. In the conclusion, a summary of the answers to the research questions is provided. Then followed with discussion of the contributions and limitation of the study. Finally, the chapter ends with the proposal for future work.

7.1 Summary

The purpose of the study was to explore the merge conflicts in an open source project by analyzing the conflicting changes that led to their occurrence and categorizing them. To achieve the purpose of the study, the following questions were answered by analyzing a sample of 40 conflicting changes extracted from the Elasticsearch project:

RQ1. How can the merge conflicts be categorized in terms of code level changes?
As describe in Section 4.3.2 and presented in Section 5.1, the merge conflicts were categorized based on the syntactic structure of the conflicting changes. In the sample dataset, six categories of the merge conflicts were identified. The categories are:

- Change of Method call or object creation (MC_OC)
- Change of an assert statement Expression (AS_EXP)
- Addition of statements in the Same area (ADD_STMT)
- Modification and removal of statements (MOD/RMV_STMT)
- Changes in Different statements in the same area (D_STMT)
- Change of IF statement condition (IF_C)

RQ2. What are the project level changes that led to the merge conflicts in an open source software project?
To understand the changes that led to the merge conflicts in the project code base, an in-depth analysis was performed in the sample. For each conflicting change, a change in the mainline branch and topic branch was analyzed by reviewing the message of the commit and its associated pull request that made the change. Eight change categories were identified from the analysis of the conflicting changes. The change categories are:

- Feature introduction
- Refactoring
- Feature enhancement

- Test improvement
- Bug fix
- Framework removal
- Breaking change fix
- Library removal

7.2 Contributions and Limitation of the Study

This thesis contributes two datasets of conflicting changes in the merge conflicts extracted from the Elasticsearch project. A first dataset contains 534 conflicting changes in all extracted merge conflicts and a second dataset contains descriptions of a sample of 40 conflicting changes. The first dataset can be used to replicate the analysis of the changes that led to the merge conflicts and categorization of the merge conflicts by selecting a sample or studying all conflicting changes. Also, can be used to validate if same data will be extracted when replicating the methodology adapted in this study. The second dataset can be used to validate the categories of the changes that led to the merge conflicts. Moreover, it may be expanded and used to answer research questions such as how many merge conflicts are due to refactoring in a project or projects, and to study relationship of changes that led to the merge conflicts.

Another contribution is the methodology that was used to analyze the conflicting changes. Researchers may adopt the methodology and improve to suit their studies which may have similar purpose with the work in this thesis. Also, since to the best of our knowledge there may be no prior work similar to this thesis work that studied merge conflicts by analyzing code base changes in the mainline branch and topic branch, which is an individual developer branch, therefore, the methodology in this thesis may provide motivation for a methodology in similar studies. Furthermore, together with the methodology, this thesis contributes the scripts used to extract the merge conflicts and conflicting changes in those merge conflicts. The datasets and scripts can be accessed online [1].

The main limitation of the study is size of the sample selected for the in-depth analysis of the conflicting changes in the merge conflicts and categorization of the merge conflicts. The size of the sample was selected due to limited time. The analysis of changes in one merge conflict on average took one day. This is because the knowledge of the subject system was required to analyze the changes in the source code and to describe those changes. Therefore, for a larger sample it would have taken longer time to analyze all conflicting changes. However, we selected the sample that is enough to represent the conflicting changes in the project under study and can be analyzed within reasonable time.

[1] https://github.com/msafirim/merge-conflicts-study

7.3 Future Work

The long-term goal of this study is to support developers when they are resolving merge conflicts by providing them with information that might reduce the perceived complexity of the merge conflicts resolution. Therefore, this study is the first step towards that goal. However, before developing a tool to help the developers, further studies are required to understand if the results in this study might hold to other projects. We would first like to conduct an in-depth analysis of the merge conflicts in more open source projects as the development history is publicly available. This will enable to acquire understanding of the merge conflicts in different development cultures and programming languages. Therefore, it will ensure good understanding of conflicting changes in the merge conflicts and may led to development of a better tool for the merge conflict resolution.

Furthermore, we would like to evaluate existing merge conflict tools to understand how they support developers when there are merge conflicts and how they prevent or reduce occurrence of the merge conflicts. The evaluation should help to understand the benefits and challenges of the existing tools and propose improvement. Also, it will allow to understand missing features in the existing tools. Moreover, the evaluation may provide insight about how a tool that support the developers during resolution of the merge conflicts can be developed.

Lastly, we propose a tool to aid the developers during the merge conflict resolution that will use information about the changes in conflicting code from version control systems. The developers may be presented with a contextual information of the changes. Similarly, this requirement was also identified by McKee et al. [8] from interviews which they conducted with developers and was ranked in the top in a survey which they asked developers to rate identified needs during conflicts resolution. Therefore, this indicate there is a need to have a tool that make use of the data from the version control systems, in particular commit messages, in a way that developers will easily view and understand what were the changes made in the other branch when they retrieve changes into their branches. Also, the tool may not be limited to providing information about the changes from other branches, it can describes the changes that were done in branches that the developers are working with. This would help in a case where a developer branch took longer time and another developer continue with it. The tool may be implemented as a plugin in an integrated development environment such as Eclipse so that developers would not switch tools when they are resolving the merge conflict. However, a decision of how to implement the tool may depend on the findings from the evaluation of merge conflict tools, which may determine how easily a merge conflict tool can be implemented. Also, the evaluation may determine possibility of extending one of the existing merge conflict tool. Then the implementation of the tool may be followed with an evaluation study. The evaluation study may be done in an experiment or a case study.

Bibliography

[1] C. Bird and T. Zimmermann, "Assessing the value of branches with what-if analysis," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 45:1–45:11.

[2] E. T. Barr, C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu, *Cohesive and Isolated Development with Branches*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 316–331.

[3] G. Gousios, M. A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 285–296.

[4] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 345–355.

[5] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive Detection of Collaboration Conflicts," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 168–178.

[6] A. Sarma, D. F. Redmiles, and A. van der Hoek, "Palantı´r: Early detection of development conflicts arising from parallel code changes," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 889–908, July 2012.

[7] B. K. Kasi and A. Sarma, "Cassandra: Proactive conflict minimization through optimized task scheduling," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 732–741.

[8] S. McKee, N. Nelson, A. Sarma, and D. Dig, "Software practitioner perspectives on merge conflicts and resolutions," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sept 2017, pp. 467–478.

[9] O. S. Initiative, "The open source definition," https://opensource.org/docs/definition.html, 2017, [Accessed Sep. 15, 2017].

[10] C. Gacek and B. Arief, "The many meanings of open source," *IEEE Software*, vol. 21, no. 1, pp. 34–40, Jan 2004.

[11] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," in *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, June 2000, pp. 263–272.

[12] M. J. Karels, "Commercializing open source software," *Queue*, vol. 1, no. 5, pp. 40:46–40:55, Jul. 2003.

[13] A. Hars and S. Ou, "Working for free? motivations for participating in open-source projects," *Int. J. Electron. Commerce*, vol. 6, no. 3, pp. 25–39, Apr. 2002.

[14] S. K. Shah, "Motivation, governance, and the viability of hybrid forms in open source software development," *Management Science*, vol. 52, no. 7, pp. 1000–1014, 2006.

[15] D. C. Gumm, "Distribution dimensions in software development projects: A taxonomy," *IEEE Software*, vol. 23, no. 5, pp. 45–51, Sept 2006.

[16] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," *Adv. Soft. Eng.*, vol. 2009, pp. 3:1–3:16, Jan. 2009.

[17] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, May 2009, pp. 1–10.

[18] B. de Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?" in *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, ser. CHASE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 36–39.

[19] Y. Saito, K. Fujiwara, H. Igaki, N. Yoshida, and H. Iida, "How do github users feel with pull-based development?" in *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, March 2016, pp. 7–11.

[20] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 358–368.

[21] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, "Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development," *Information and Software Technology*, vol. 84, no. Supplement C, pp. 48 – 62, 2017.

[22] G. Robles and J. M. González-Barahona, *A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–14.

[23] L. Nyman and T. Mikkonen, *To Fork or Not to Fork: Fork Motivations in SourceForge Projects*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 259–268.

[24] Ş. Stănciulescu, S. Schulze, and A. Wąsowski, "Forked and integrated variants in an open-source firmware project," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sept 2015, pp. 151–160.

[25] S. Phillips, J. Sillito, and R. Walker, "Branching and merging: An investigation into current version control practices," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '11. New York, NY, USA: ACM, 2011, pp. 9–15.

[26] S. Chacon, *Pro Git*, 1st ed. Apress, 2009.

[27] C. Walrad and D. Strom, "The importance of branching models in scm," *Computer*, vol. 35, no. 9, pp. 31–38, Sep 2002.

[28] A. Nieminen, "Real-time collaborative resolving of merge conflicts," in *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, Oct. 2012, pp. 540–543.

[29] T. Mens, "A state-of-the-art survey on software merging," *IEEE Transactions on Software Engineering*, vol. 28, no. 5, pp. 449–462, May 2002.

[30] P. Accioly, P. Borba, and G. Cavalcanti, "Understanding semi-structured merge conflict characteristics in open-source java projects," *Empirical Software Engineering*, Dec 2017.

[31] S. Apel, J. Liebig, B. Brandl, C. Lengauer, and C. Kästner, "Semistructured merge: Rethinking merge in revision control systems," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. ACM, 2011, pp. 190–200.

[32] G. G. L. de Menezes, "On the nature of software merge conflicts," Ph.D. dissertation, Federal Fluminense University, Dec 2016, [Accessed Jan. 09, 2017].

[33] M. L. Guimarães and A. R. Silva, "Improving early detection of software merge conflicts," in *2012 34th International Conference on Software Engineering (ICSE)*, Jun. 2012, pp. 342–352.

[34] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[35] Elastic, "Getting started," https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html, 2017, [Accessed Aug. 16, 2017].

[36] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, Jul 1999.

[37] R. K. Yin, *Case study research: Design and methods*, 5th ed. Sage publications, 2014.

[38] S. Apel, O. Leßenich, and C. Lengauer, "Structured merge with auto-tuning: Balancing precision and performance," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 120–129.

[39] A. E. Hassan, "Predicting faults using the complexity of code changes," in *2009 IEEE 31st International Conference on Software Engineering*, May 2009, pp. 78–88.

Appendices

A

Descriptions of the Features Studied in the Prestudy

CancelableSearchAction

Cancels long running searches using standard task cancellation mechanism. For example:

```
$ curl -XPOST 'localhost:9200/_tasks/task_id:1/_cancel?pretty'
```

CustomizableShardWeights

Enables balancing of shard weights by distributing the load evenly in the cluster. It allows individual shards in a cluster that are often queried to be more evenly distributed across nodes. The shards are supposed to have a high weight manually assigned to them, so that Elasticsearch can distribute them accordingly, providing better load balancing by spreading high-load shards more than low-load ones.

FunctionScoreQueryFunctions

Provides the option to use several scores of queries with mathematical functions and allows to wrap them in one query. It aggregates filter and sort results in a search.

GeoHeatMapAggregator

Visualize data based on geographical location as heat map by dividing the map into grids, and grouping points into buckets. It allows a user to take points (x and y coordinates) from a map or a graph and get a "heat map" of where most points were found.

GeoHeatMapGridAggregator

This feature extends GeoHeatMapAggregator feature by setting the maximum allowable error specified as fraction of the shape size when determining where an indexed shape is relative to the heatmap cells.

GeoMeansAggregator

Performs k-means clustering on a GeoPoint field.

SearchIngestProcessor

Returns the most relevant hit by running a search query against an Elasticsearch cluster. The Elasticsearch cluster can be internal or external. It uses REST layer to communicate. The search request is sent with size 1 in order to get only data from the more relevant hit.

NodesUsageAPI-REST

Provides statistics of node usage based on user actions. It allows to see how many times each rest action has been called on a node.

ScrollPersistence

Makes scroll to persist across clusters since the scroll is lost when the node where is stored shut down or moved. This means the order of the results can change when another page in a result set will be requested.

StandardNumberAnalysis (Plugin)

Finds standard numbers and index them in canonical form with their valid variants. The canonical form must be deduced from an input so that standard numbers can be compared for equivalence since they may appear in some literal variants.

StringFieldsAnalyzerProcessor

Tokenize search strings to yield more relevant searches and results.

SynonymsGraph

Provides search result that extends further than words in the search query by also including words with similar meaning. It extends a synonym feature to also include multiword queries and results.

TermsWithoutGlobalOrdinals

Helps to optimize the performance when running terms aggregation by leveraging map ordinals. Since there is no option for rebuilding global ordinals, this feature provides alternative for users who have issues with global ordinals building.

TopHitsAggregationSorting

Sorts terms bucket by using score assigned by a previous query in the same search request.

UnifiedHighlighter

Combines multiple highlighting modes. It includes plain mode that analyzes the plain text directly, postings mode that uses the postings offsets to perform the highlight, and fast vector highlighter mode that uses the term vectors to perform the highlighting. It allows more advanced highlighting in regard to a specific task that a user wants to perform.

xmlFormat

Allows users to request results in XML instead of JSON format

B

Dataset of changes that led to the Merge Conflicts

Appendix B.1. Extracted commits and files associated with the merge conflicts in the dataset

Table B.1 shows the commits and file path associated with each merge conflict in the dataset. The commits include merge commit and its associated parent commits, common ancestor commit used when the changes were merged, and change commit in which the change that led to the merge conflict was made.

Table B.1: Commits and files associated with the merge conflicts

Conflict #	Merge Commit	Common Ancestor Commit	First Merge Parent	Second Merge Parent	File Path	Change Commit	
						Mainline	Branch
1	5717ac3c	f217eb8a	210e101f	8758c541	core/src/test/java/org/elasticsearch/search/aggregations/metrics/min/InternalMinTests.java	e71b26f4	75fdc944
2	d3417fb0	615928e8	9ceb0f2c	179dd885	core/src/main/java/org/elasticsearch/action/delete/TransportDeleteAction.java	63c07282	5fb0f9a8
3	d3417fb0	615928e8	9ceb0f2c	179dd885	core/src/main/java/org/elasticsearch/index/engine/InternalEngine.java	fa3ee6b9	3106948a
4	d3417fb0	615928e8	9ceb0f2c	179dd885	core/src/test/java/org/elasticsearch/index/IndexModuleTests.java	6418f89f	48443259
5	d3417fb0	615928e8	9ceb0f2c	179dd885	core/src/test/java/org/elasticsearch/index/engine/InternalEngineTests.java	1587a77f	5fb0f9a8
6	d3417fb0	615928e8	9ceb0f2c	179dd885	core/src/test/java/org/elasticsearch/index/engine/InternalEngineTests.java	1587a77f	5fb0f9a8
7	25fd9e26	74af0e36	c809671e	85402d52	core/src/main/java/org/elasticsearch/index/shard/TranslogRecoveryPerformer.java	a0becd26	5fb0f9a8
8	25fd9e26	74af0e36	c809671e	85402d52	core/src/test/java/org/elasticsearch/index/translog/TranslogTests.java	a0becd26	5fb0f9a8
9	d0765d07	9a322710	9884b7dc	74af0e36	core/src/main/java/org/elasticsearch/action/support/replication/TransportReplicationAction.java	b4064ce4	48443259
10	112669da	be168f52	275ea683	0cae9ad3	core/src/main/java/org/elasticsearch/index/engine/InternalEngine.java	4f49a261	3106948a
11	bbd5f26d	0eb1a816	6380560d	d55f719f	core/src/main/java/org/elasticsearch/common/settings/Settings.java	ba14aca2	3adaf096
12	15d3d744	6c15e782	dcd2642d	d3d57da8	core/src/test/java/org/elasticsearch/index/engine/ShadowEngineTests.java	5d8f6843	5fb0f9a8
13	27d4994a	4bfef1fd	2c6e78e1	5e8656af	core/src/main/java/org/elasticsearch/action/admin/cluster/stats/ClusterStatsNodes.java	ce86fc56	2c6e78e1
14	bf390a93	675d940f	da199224	4ac4f3c8	core/src/main/java/org/elasticsearch/cluster/node/DiscoveryNodeService.java	cd12241e	eb941d80
15	bd96075f	22ee910f	ce6ec511	e4bed0c9	core/src/main/java/org/elasticsearch/action/admin/cluster/node/stats/NodeStats.java	31740e27	121e7c8c
16	69c83b34	c9e1ccf6	e7cffa5e	b4db26ea	core/src/main/java/org/elasticsearch/search/suggest/SuggestParseElement.java	e72dac91	9e0f6e3f
17	c11cf3bf	82567f1b	e4031932	29e34439	core/src/main/java/org/elasticsearch/common/settings/Setting.java	52acf0e6	31b5e088
18	4bb5b410	77dbfbc9	4d0feff2	b5aee207	core/src/main/java/org/elasticsearch/action/admin/cluster/stats/TransportClusterStatsAction.java	7835525f	3106948a
19	4bb5b410	77dbfbc9	4d0feff2	b5aee207	core/src/test/java/org/elasticsearch/action/support/broadcast/node/TransportBroadcastByNodeActionTests.java	2a137b55	b364cf54
20	4bb5b410	77dbfbc9	4d0feff2	b5aee207	core/src/test/java/org/elasticsearch/cluster/routing/allocation/decider/DiskThresholdDeciderTests.java	4937531a	b364cf54
21	4bb5b410	77dbfbc9	4d0feff2	b5aee207	core/src/test/java/org/elasticsearch/cluster/routing/allocation/decider/DiskThresholdDeciderUnitTests.java	2a137b55	b364cf54
22	cd8320b1	3d98756e	e6f9cbce	ec31feca	core/src/main/java/org/elasticsearch/search/aggregations/bucket/filters/FiltersAggregator.java	cc41e6e7	6df27fe0
23	83a5fe96	7a469538	31afc8a9	d0a10b33	core/src/test/java/org/elasticsearch/gateway/ReplicaShardAllocatorTests.java	3a442db9	b364cf54
24	27d8509f	5d001d15	99e328c9	7bca97bb	core/src/main/java/org/elasticsearch/index/shard/IndexShard.java	1bff08b2	6ae8ca9a
25	27d8509f	5d001d15	99e328c9	7bca97bb	core/src/test/java/org/elasticsearch/indices/memory/IndexingMemoryControllerTests.java	5341404f	f27c0adb
26	503a166b	b4e0c876	a8382de0	afcaa593	core/src/main/java/org/elasticsearch/action/delete/DeleteResponse.java	fafeb3ab	a2cda4e3
27	68f1a87c	54022774	95e8a39b	fafeb3ab	core/src/main/java/org/elasticsearch/action/support/replication/TransportReplicationAction.java	025e9818	5fb0f9a8
28	68f1a87c	54022774	95e8a39b	fafeb3ab	core/src/main/java/org/elasticsearch/action/update/UpdateHelper.java	fafeb3ab	5fb0f9a8
29	68f1a87c	54022774	95e8a39b	fafeb3ab	core/src/main/java/org/elasticsearch/rest/action/update/RestUpdateAction.java	fafeb3ab	5fb0f9a8
30	95e8a39b	c512cc52	bffd55dd	54022774	core/src/main/java/org/elasticsearch/cluster/ClusterState.java	fef043a5	b364cf54
31	95e8a39b	c512cc52	bffd55dd	54022774	core/src/main/java/org/elasticsearch/cluster/metadata/IndexMetaData.java	fef043a5	b364cf54
32	d68b8101	18a75fb3	1e5af7b6	6a2fa73f	core/src/test/java/org/elasticsearch/cluster/routing/allocation/PrimaryElectionRoutingTests.java	e31d66d1	b364cf54
33	ba68a8df	7b74f0dd	672a54b3	3ab39385	core/src/main/java/org/elasticsearch/index/IndexService.java	c0eca94a	46d10f1b
34	1228a9fe	542cf66c	197313c1	c8d1f7aa	core/src/test/java/org/elasticsearch/percolator/PercolatorBackwardsCompatibilityIT.java	effaaf05	904cbf53
35	73f7df51	c10f116a	d49a744b	2c618a11	core/src/main/java/org/elasticsearch/index/query/HasParentQueryBuilder.java	ab0847e0	18bec264
36	73f7df51	c10f116a	d49a744b	2c618a11	core/src/test/java/org/elasticsearch/search/child/ChildQuerySearchIT.java	ab0847e0	18bec264
37	1b8047e5	35f9ee7a	80b59e0d	5ae00a61	core/src/main/java/org/elasticsearch/discovery/local/LocalDiscovery.java	14e48824	80b59e0d
38	f3d63095	dcf3f467	be638fb6	4010e7e9	core/src/test/java/org/elasticsearch/index/analysis/synonyms/SynonymsAnalysisTest.java	6dcfda99	be638fb6
39	654dc208	ed561cd4	10f80167	bbaf4710	core/src/main/java/org/elasticsearch/common/io/stream/StreamInput.java	e7eb9cf4	ff9041dc
40	9171ff09	cdd13253	33668a8d	f4a143d1	core/src/main/java/org/elasticsearch/index/query/PrefixQueryBuilder.java	0a526be3	283fe90e

Appendix B.2. Changes that led to Merge Conflicts

B.2.1 Conflict 1

Category: Change of Method call or object creation

Mainline Change

Category: Test improvement

Project level perspective. Some aggregations tests were changed to use correct document field datatypes. The change was made to not test metrics aggregations (such as min or max) that expect a numeric value with a document field datatype such as IP (IPv4 and IPv6 addresses). The aggregations tests were modified so that not to return wrong aggregations which are also used in other tests.

Code level perspective. Listing below shows an example of a change in aggregations tests to use correct document field datatypes. In the example, a parameter that return random selected `DocValueFormat` (document field datatype) was replaced with a method `randomNumericDocValueFormat` that return random selected format from a list of datatype formats. The list of datatype formats contains Raw, Geohash, and IP datatypes.

```
     protected InternalMin createTestInstance(String name, List<PipelineAggregator
         ↪ > pipelineAggregators, Map<String, Object> metaData) {
-        return new InternalMin(name, randomDouble(),
-                randomFrom(DocValueFormat.BOOLEAN, DocValueFormat.GEOHASH,
    ↪ DocValueFormat.IP, DocValueFormat.RAW), pipelineAggregators,
-                metaData);
+        return new InternalMin(name, randomDouble(), randomNumericDocValueFormat
    ↪ (), pipelineAggregators, metaData);
     }
```

Branch Change

Category: Feature introduction

Project level perspective. Parsing from xContent, which is an abstraction on top of content such as Json, was added to some of the aggregations. The aggregations are max, min, avg, sum and value count. The change was part of high level REST client aggregations parsing feature implementation.

Code level perspective. Listing below shows a change due to addition of xContent parsing to metrics aggregations. In the listing, test parameters for document's field value and format were modified. The document's field value was improved to randomly select infinity values. On the other hand, the document's field format was updated to remove geohash format and replace with decimal format.

```
     protected InternalMin createTestInstance(String name, List<PipelineAggregator
         ↪ > pipelineAggregators, Map<String, Object> metaData) {
-        return new InternalMin(name, randomDouble(),
-                randomFrom(DocValueFormat.BOOLEAN, DocValueFormat.GEOHASH,
    ↪ DocValueFormat.IP, DocValueFormat.RAW), pipelineAggregators,
```

```
-                   metaData);
+           double value = frequently() ? randomDouble() : randomFrom(new Double[] {
    ↪ Double.NEGATIVE_INFINITY, Double.POSITIVE_INFINITY });
+           DocValueFormat formatter = randomFrom(new DocValueFormat.Decimal("#####"
    ↪ ), DocValueFormat.BOOLEAN, DocValueFormat.RAW);
+           return new InternalMin(name, value, formatter, pipelineAggregators,
    ↪ metaData);
     }
```

B.2.2 Conflict 2

Category: Change of Method call or object creation, Addition of statements in the Same area

Mainline Change

Category: Refactoring

Project level perspective. There are two changes from a common ancestor version that led to the merge conflict. In the first change, handling of write operation such as delete was simplified in transport actions that modify data in shards. With this change, failures occurred before executing engine's write operations are conveyed through a failure operation type. The failure operation type can be request failure such as document version conflict, transient operation failure such as when initializing shard, and environment failure such as when there is out of disk error. This change was part of an enhancement to handle failure types appropriately since there was no distinction between environment and request failures for the write operation.
In the second change, responsibilities to update document's version and version type in a shard bulk request to shard replicas after delete operation on primary shards were moved to a caller of the operation. Before this change, update of the request version and version type on the shard replicas was done after execution of delete operation on the primary shard. This change was made to ensure an execution of write operation, that is, deletion of shards does not have side effects.

Code level perspective. In order to handle the failures occurred before executing engine's write operations appropriately, the return type of method `executeDeleteRequestOnPrimary` was changed from `WriteResult<DeleteResponse>` to `Engine.DeleteResult` so that to distinguish failures occurred due to delete operation with engine level. In the second change, to ensure the execution of write operation does not have side effects, a `delete` variable which holds preparation result of delete operation on primary shard was made final and response of the delete operation was returned to the operation caller. Furthermore, the request update responsibility was moved to the operation caller.

```
-    public static WriteResult<DeleteResponse> executeDeleteRequestOnPrimary(
    ↪ DeleteRequest request, IndexShard indexShard) {
-        Engine.Delete delete = indexShard.prepareDeleteOnPrimary(request.type(),
    ↪ request.id(), request.version(), request.versionType());
-        indexShard.delete(delete);
-        // update the request with the version so it will go to the replicas
-        request.versionType(delete.versionType().
    ↪ versionTypeForReplicationAndRecovery());
```

```
-         request.version(delete.version());
-
-         assert request.versionType().validateVersionForWrites(request.version());
-         DeleteResponse response = new DeleteResponse(indexShard.shardId(),
    ↪ request.type(), request.id(), delete.version(), delete.found());
-         return new WriteResult<>(response, delete.getTranslogLocation());
+     public static Engine.DeleteResult executeDeleteRequestOnPrimary(DeleteRequest
    ↪  request, IndexShard primary) {
+         final Engine.Delete delete = primary.prepareDeleteOnPrimary(request.type
    ↪ (), request.id(), request.version(), request.versionType());
+         return primary.delete(delete);
      }
```

Branch Change

Category: Feature introduction

Project level perspective. A counter was added to each write operation (such as indexing) on a shard to enforce semantics of primary terms. The primary terms track number of times when a new primary shard is selected among existing replica shards after an old primary shard failed. The primary terms help to identify operations from the old failed primary shard. With this change, the counter was added to identify the write operations from the failed primary shard so that other shards should not execute the operations received from the failed primary shard. This is an enhancement to support implementation of write operations sequence numbers feature. The feature orders operations on shards against each other. Before the implementation of the feature, the ordering of operations was done on a per document basis and replicated to replica shards after write operations were applied on the primary shard. The ordering of operations on a per shard basis enable implementations of high level features such as changes API which allow to follow changes made to documents in the shard and index.

Code level perspective. To enforce semantics of the primary terms, a counter `delete.seqNo()` for delete operation on a primary shard was added. Then, the counter was passed to the sequence number assigned to `delete` request on the primary shard. Furthermore, the `delete.seqNo()` counter was passed to the response of delete action.

```
     public static WriteResult<DeleteResponse> executeDeleteRequestOnPrimary(
         ↪ DeleteRequest request, IndexShard indexShard) {
         Engine.Delete delete = indexShard.prepareDeleteOnPrimary(request.type(),
             ↪ request.id(), request.version(), request.versionType());
         indexShard.delete(delete);
        // update the request with the version so it will go to the replicas
         request.versionType(delete.versionType().
             ↪ versionTypeForReplicationAndRecovery());
         request.version(delete.version());
+        request.seqNo(delete.seqNo());

         assert request.versionType().validateVersionForWrites(request.version());
-        DeleteResponse response = new DeleteResponse(indexShard.shardId(),
    ↪ request.type(), request.id(), delete.version(), delete.found());
+        DeleteResponse response = new DeleteResponse(indexShard.shardId(),
    ↪ request.type(), request.id(), delete.seqNo(), delete.version(), delete.
    ↪ found());
         return new WriteResult<>(response, delete.getTranslogLocation());
     }
```

B.2.3 Conflict 3

Category: Addition of statements in the Same area

Mainline Change

Category: Refactoring

Project level perspective. Same as first change in B.2.2, which simplified handling of write operations failure.

Code level perspective. To handle failure in case there is document's version conflict while recovering from `delete` operation, the execution of the `delete` operation is skipped and result of the operation `deleteResult` is updated to the `delete.version()` version after the `delete` operation.

```
-    private void innerDelete(Delete delete) throws IOException {
+    private DeleteResult innerDelete(Delete delete) throws IOException {
      ...
             final long expectedVersion = delete.version();
-            if (checkVersionConflict(delete, currentVersion, expectedVersion,
    ↪ deleted)) return;
-            final long updatedVersion = updateVersion(delete, currentVersion,
    ↪ expectedVersion);
-            final boolean found = deleteIfFound(delete, currentVersion, deleted,
    ↪ versionValue);
-            delete.updateVersion(updatedVersion, found);
-            maybeAddToTranslog(delete, updatedVersion, Translog.Delete::new,
    ↪ DeleteVersionValue::new);
+            final DeleteResult deleteResult;
+            if (checkVersionConflict(delete, currentVersion, expectedVersion,
    ↪ deleted)) {
+                // skip executing delete because of version conflict on recovery
+                deleteResult = new DeleteResult(expectedVersion, true);
+            } else {
+                updatedVersion = delete.versionType().updateVersion(
    ↪ currentVersion, expectedVersion);
+                found = deleteIfFound(delete.uid(), currentVersion, deleted,
    ↪ versionValue);
+                deleteResult = new DeleteResult(updatedVersion, found);
+                location = delete.origin() != Operation.Origin.
    ↪ LOCAL_TRANSLOG_RECOVERY
+                        ? translog.add(new Translog.Delete(delete, deleteResult))
+                        : null;
+                versionMap.putUnderLock(delete.uid().bytes(),
+                        new DeleteVersionValue(updatedVersion, engineConfig.
    ↪ getThreadPool().estimatedTimeInMillis()));
+                deleteResult.setTranslogLocation(location);
+            }
+            deleteResult.setTook(System.nanoTime() - delete.startTime());
+            deleteResult.freeze();
+            return deleteResult;
         }
     }
```

Branch Change

Category: Feature introduction

Project level perspective. Local checkpoints were introduced on a shard level. A local checkpoint is a last sequence number after all previous operations were processed.

The sequence numbers are incremented for each operation on the shard. Therefore, after previous operations were completed which have lower sequence numbers, the local checkpoint will have highest sequence number. The checkpoints were introduced to support implementation of write operations sequence numbers feature (see also B.2.2).

Code level perspective. As part of introduction of local checkpoints (highest sequence numbers on shard level), a `delete.seqNo()` sequence number for delete operation is passed to `markSeqNoAsCompleted` method which marks the sequence number as completed. The sequence number `delete.seqNo()` needs to be assigned to a shard before marked as completed.

```
      private void innerDelete(Delete delete) throws IOException {
          ...
              final long expectedVersion = delete.version();
              if (checkVersionConflict(delete, currentVersion, expectedVersion,
                  ↪ deleted)) return;

+             maybeUpdateSequenceNumber(delete);
              final long updatedVersion = updateVersion(delete, currentVersion,
                  ↪ expectedVersion);
-
              final boolean found = deleteIfFound(delete, currentVersion, deleted,
                  ↪ versionValue);
-
              delete.updateVersion(updatedVersion, found);

              maybeAddToTranslog(delete, updatedVersion, Translog.Delete::new,
                  ↪ DeleteVersionValue::new);
+         } finally {
+             if (delete.seqNo() != SequenceNumbersService.UNASSIGNED_SEQ_NO) {
+                 seqNoService.markSeqNoAsCompleted(delete.seqNo());
+             }
          }
      }
```

B.2.4 Conflict 4

Category: Change of Method call or object creation

Mainline Change

Category: Framework removal

Project level perspective. Provider of node services that hold index and node level services used by a shard was removed. The change was part of removing Guice[1] dependency injection framework in the index level. After the change, the node services are provided were they are needed.

Code level perspective. As part of node services provider removal in the index level, tests on index module were modified. The `nodeServicesProvider` parameter was removed in the `newIndexService` method which create new index service with custom implementations.

```
    public void testWrapperIsBound() throws IOException {
```

[1]https://github.com/google/guice

```
        IndexModule module = new IndexModule(indexSettings, null, new
            ↪ AnalysisRegistry(environment, emptyMap(), emptyMap(), emptyMap(),
            ↪ emptyMap()));
        module.setSearcherWrapper((s) -> new Wrapper());
        module.engineFactory.set(new MockEngineFactory(AssertingDirectoryReader.
            ↪ class));
-       IndexService indexService = module.newIndexService(nodeEnvironment,
    ↪ deleter, nodeServicesProvider, indicesQueryCache, mapperRegistry, new
    ↪ IndicesFieldDataCache(settings, listener));
+       IndexService indexService = newIndexService(module);
        assertTrue(indexService.getSearcherWrapper() instanceof Wrapper);
        assertSame(indexService.getEngineFactory(), module.engineFactory.get());
        indexService.close("simon says", false);
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Global checkpoints were introduced to represent common part of a history across shard replicas at a given time. The global checkpoints are updated by a primary shard. Similar to local checkpoints B.2.3, a global checkpoint is a last sequence number after all previous operations were processed across shard replicas.

Code level perspective. Listing below shows a change in index module tests as part of global checkpoints introduction. A parameter `shardId -> {}` was added when customizing index level services. The parameter injects shard ID to shard level components.

```
    public void testWrapperIsBound() throws IOException {
        IndexModule module = new IndexModule(indexSettings, null, new
            ↪ AnalysisRegistry(environment, emptyMap(), emptyMap(), emptyMap(),
            ↪ emptyMap()));
        module.setSearcherWrapper((s) -> new Wrapper());
        module.engineFactory.set(new MockEngineFactory(AssertingDirectoryReader.
            ↪ class));
-       IndexService indexService = module.newIndexService(nodeEnvironment,
    ↪ deleter, nodeServicesProvider, indicesQueryCache, mapperRegistry, new
    ↪ IndicesFieldDataCache(settings, listener));
+       IndexService indexService = module.newIndexService(nodeEnvironment,
    ↪ deleter, nodeServicesProvider, indicesQueryCache,
+               mapperRegistry, shardId -> {} ,new IndicesFieldDataCache(settings,
    ↪  listener));
        assertTrue(indexService.getSearcherWrapper() instanceof Wrapper);
        assertSame(indexService.getEngineFactory(), module.engineFactory.get());
        indexService.close("simon says", false);
    }
```

B.2.5 Conflict 5

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Index operation on shard level was made immutable. To make the index operation immutable, a result of the index operation is logged before returning response. The result contains the following data; transaction log location, index version, index status, index time, and index estimated size. This change was part of simplifying handling of write operations failure (see B.2.2).

Code level perspective. Listing below shows changes as part of making index operation immutable. A test on versioning new index compares primary index and its replica. As part of making index operation immutable, a variable `indexResult` is assigned to result of indexing `index`.

```
     public void testVersioningNewIndex() {
         ParsedDocument doc = testParsedDocument("1", "1", "test", null, -1, -1,
             ↪ testDocument(), B_1, null);
         Engine.Index index = new Engine.Index(newUid("1"), doc);
-        engine.index(index);
-        assertThat(index.version(), equalTo(1L));
+        Engine.IndexResult indexResult = engine.index(index);
+        assertThat(indexResult.getVersion(), equalTo(1L));

-        index = new Engine.Index(newUid("1"), doc, index.version(), index.
    ↪ versionType().versionTypeForReplicationAndRecovery(), REPLICA, 0, -1, false
    ↪ );
-        replicaEngine.index(index);
-        assertThat(index.version(), equalTo(1L));
+        index = new Engine.Index(newUid("1"), doc, indexResult.getVersion(),
    ↪ index.versionType().versionTypeForReplicationAndRecovery(), REPLICA, 0, -1,
    ↪  false);
+        indexResult = replicaEngine.index(index);
+        assertThat(indexResult.getVersion(), equalTo(1L));
     }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2.

Code level perspective. Listing below shows a change to enforce semantics of the primary terms. A counter `index.seqNo()` for index operation on a primary shard was passed as a parameter when adding an index replica `index`.

```
     public void testVersioningNewIndex() {
         ParsedDocument doc = testParsedDocument("1", "1", "test", null, -1, -1,
             ↪ testDocument(), B_1, null);
         Engine.Index index = new Engine.Index(newUid("1"), doc);
         engine.index(index);
         assertThat(index.version(), equalTo(1L));

-        index = new Engine.Index(newUid("1"), doc, index.version(), index.
    ↪ versionType().versionTypeForReplicationAndRecovery(), REPLICA, 0, -1, false
    ↪ );
+        index = new Engine.Index(newUid("1"), doc, index.seqNo(), index.version()
    ↪ , index.versionType().versionTypeForReplicationAndRecovery(), REPLICA, 0,
    ↪ -1, false);
         replicaEngine.index(index);
         assertThat(index.version(), equalTo(1L));
     }
```

B.2.6 Conflict 6

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Same as B.2.5.

Code level perspective. Listing below shows a test to replay transaction log after `index` operation failure. To make index operation immutable, result of indexing indexing operation is assigned to a variable `indexResult`.

```
      public void testTranslogReplayWithFailure() throws IOException {
        final int numDocs = randomIntBetween(1, 10);
        for (int i = 0; i < numDocs; i++) {
            ParsedDocument doc = testParsedDocument(Integer.toString(i), Integer.
                ↪ toString(i), "test", null, -1, -1, testDocument(), new
                ↪ BytesArray("{}"), null);
            Engine.Index firstIndexRequest = new Engine.Index(newUid(Integer.
                ↪ toString(i)), doc, Versions.MATCH_DELETED, VersionType.
                ↪ INTERNAL, PRIMARY, System.nanoTime(), -1, false);
-           engine.index(firstIndexRequest);
-           assertThat(firstIndexRequest.version(), equalTo(1L));
+           Engine.IndexResult indexResult = engine.index(firstIndexRequest);
+           assertThat(indexResult.getVersion(), equalTo(1L));
        }
        ...
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2.

Code level perspective. In the listing below, a counter `SequenceNumbersService.UNASSIGNED_SEQ_NO`, which track unassigned primary terms due to failure, was passed as parameter when creating an index. The code tests replaying transaction log after `index` operation failure.

```
      public void testTranslogReplayWithFailure() throws IOException {
        final int numDocs = randomIntBetween(1, 10);
        for (int i = 0; i < numDocs; i++) {
            ParsedDocument doc = testParsedDocument(Integer.toString(i), Integer.
                ↪ toString(i), "test", null, -1, -1, testDocument(), new
                ↪ BytesArray("{}"), null);
-           Engine.Index firstIndexRequest = new Engine.Index(newUid(Integer.
    ↪ toString(i)), doc, Versions.MATCH_DELETED, VersionType.INTERNAL, PRIMARY,
    ↪ System.nanoTime(), -1, false);
+           Engine.Index firstIndexRequest = new Engine.Index(newUid(Integer.
    ↪ toString(i)), doc, SequenceNumbersService.UNASSIGNED_SEQ_NO, Versions.
    ↪ MATCH_DELETED, VersionType.INTERNAL, PRIMARY, System.nanoTime(), -1, false)
    ↪ ;
            engine.index(firstIndexRequest);
            assertThat(firstIndexRequest.version(), equalTo(1L));
        }
        ...
    }
```

B.2.7 Conflict 7

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. Adding documents with autogenerated IDs (IDs generated automatically by Elasticsearch during data indexing) to the index was optimized for append-only use case to improve performance. One of the append-only use case is when indexing operation is not completed and a request is sent again since a sending node did not receive response whether the operation was successful or not. This can lead to duplicate requests in the receiving node (node with primary shard). To optimize the request in this case, a timestamp was added for each request to allow identification of failed requests. The timestamp allows to compare the retried request's timestamp with engine timestamp and if the engine timestamp is lower than request timestamp, it means there is no retried request with the same timestamp that has been run before. Therefore, it is safe for engine to execute the request which is optimized for performance.

Code level perspective. Listing below shows a change which is part of optimizing documents indexing with the autogenerated ID for append only case. A parameter `index.getAutoGeneratedIdTimestamp()` was added in a method `prepareIndex` which prepare document for indexing.

```
    private void performRecoveryOperation(Engine engine, Translog.Operation
        ↪ operation, boolean allowMappingUpdates, Engine.Operation.Origin origin
        ↪ ) {
+
        try {
            switch (operation.opType()) {
                case INDEX:
                    ...
                    Engine.Index engineIndex = IndexShard.prepareIndex(docMapper(
                        ↪ index.type()), source(shardId.getIndexName(), index.
                        ↪ type(), index.id(), index.source())
                            .routing(index.routing()).parent(index.parent()).
                                ↪ timestamp(index.timestamp()).ttl(index.ttl()),
-                           index.version(), index.versionType().
    ↪ versionTypeForReplicationAndRecovery(), origin);
+                           index.version(), index.versionType().
    ↪ versionTypeForReplicationAndRecovery(), origin, index.
    ↪ getAutoGeneratedIdTimestamp(), true);
                    maybeAddMappingUpdate(engineIndex.type(), engineIndex.
                        ↪ parsedDoc().dynamicMappingsUpdate(), engineIndex.id(),
                        ↪  allowMappingUpdates);
                ...
            }
        ...
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2.

Code level perspective. Listing below shows a change to enforce semantic of primary terms. A counter `index.seqNo()` for index operation was added as parameter in method `prepareIndex`, which prepares `index` for indexing.

```
    private void performRecoveryOperation(Engine engine, Translog.Operation
        ↪ operation, boolean allowMappingUpdates, Engine.Operation.Origin origin
        ↪ ) {
        try {
            switch (operation.opType()) {
                case INDEX:
                    Translog.Index index = (Translog.Index) operation;
                    Engine.Index engineIndex = IndexShard.prepareIndex(docMapper(
                        ↪ index.type()), source(shardId.getIndexName(), index.
                        ↪ type(), index.id(), index.source())
-                           .routing(index.routing()).parent(index.parent()).
    ↪ timestamp(index.timestamp()).ttl(index.ttl()),
+                           .routing(index.routing()).parent(index.parent()).
    ↪ timestamp(index.timestamp()).ttl(index.ttl()), index.seqNo(),
                            index.version(), index.versionType().
                                ↪ versionTypeForReplicationAndRecovery(), origin);
                    maybeAddMappingUpdate(engineIndex.type(), engineIndex.
                        ↪ parsedDoc().dynamicMappingsUpdate(), engineIndex.id(),
                        ↪  allowMappingUpdates);
                    if (logger.isTraceEnabled()) {
                    ...
            }
        ...
    }
```

B.2.8 Conflict 8

Category: Change of an assert statement Expression

Mainline Change

Category: Feature enhancement

Project level perspective. Same as B.2.7.

Code level perspective. Listing below shows a change as part of optimizing documents indexing with the autogenerated ID for append only case. In the listing, an exception message `ex.getMessage()` was updated to reflect optimization of the documents indexing.

```
    public void testRecoveryUncommittedCorruptedCheckpoint() throws IOException {
        ...
        try (Translog translog = new Translog(config, translogGeneration)) {
            fail("corrupted");
        } catch (IllegalStateException ex) {
-           assertEquals(ex.getMessage(), "Checkpoint file translog-2.ckp already
    ↪  exists but has corrupted content expected: Checkpoint{offset=2683, numOps
    ↪ =55, translogFileGeneration= 2} but got: Checkpoint{offset=0, numOps=0,
    ↪ translogFileGeneration= 0}");
+           assertEquals(ex.getMessage(), "Checkpoint file translog-2.ckp already
    ↪  exists but has corrupted content expected: Checkpoint{offset=3123, numOps
    ↪ =55, translogFileGeneration= 2} but got: Checkpoint{offset=0, numOps=0,
    ↪ translogFileGeneration= 0}");
        }
```

```
        Checkpoint.write(FileChannel::open, config.getTranslogPath().resolve(
            ↪ Translog.getCommitCheckpointFileName(read.generation)), read,
            ↪ StandardOpenOption.WRITE, StandardOpenOption.TRUNCATE_EXISTING);
        try (Translog translog = new Translog(config, translogGeneration)) {
        ...
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2.

Code level perspective. As part of a change to enforce semantics of primary terms, an exception message `ex.getMessage()` was updated when testing recovery of corrupted translogs that were not committed due to some cases such as user error. In the message, an offset is updated to reflect changes added to enforce semantics of the primary terms.

```
    public void testRecoveryUncommittedCorruptedCheckpoint() throws IOException {
        ...
        try (Translog translog = new Translog(config, translogGeneration)) {
            fail("corrupted");
        } catch (IllegalStateException ex) {
-           assertEquals(ex.getMessage(), "Checkpoint file translog-2.ckp already
    ↪  exists but has corrupted content expected: Checkpoint{offset=2683, numOps
    ↪ =55, translogFileGeneration= 2} but got: Checkpoint{offset=0, numOps=0,
    ↪ translogFileGeneration= 0}");
+           assertEquals(ex.getMessage(), "Checkpoint file translog-2.ckp already
    ↪  exists but has corrupted content expected: Checkpoint{offset=2738, numOps
    ↪ =55, translogFileGeneration= 2} but got: Checkpoint{offset=0, numOps=0,
    ↪ translogFileGeneration= 0}");
        }
        Checkpoint.write(FileChannel::open, config.getTranslogPath().resolve(
            ↪ Translog.getCommitCheckpointFileName(read.generation)), read,
            ↪ StandardOpenOption.WRITE, StandardOpenOption.TRUNCATE_EXISTING);
        try (Translog translog = new Translog(config, translogGeneration)) {
        ...
    }
```

B.2.9 Conflict 9

Category: Modification and removal of statements

Mainline Change

Category: Bug fix

Project level perspective. The blocking of indexing operations during primary shard relocation was modified to put operations which cannot be executed during the relocation phase on a queue. The operations on the queue will be executed once the primary shard relocation is completed. This change was made to fix a situation that can lead to a deadlock when the primary shard relocation and indexing operations on the primary shard occurs concurrently.

Code level perspective. Listing below shows changes which are part of fixing the deadlock situation during primary shard relocation. The replication of indexed `result` to the replica shard was removed during the primary shard relocation.

```
         protected void doRun() throws Exception {
             setPhase(task, "replica");
             assert request.shardId() != null : "request shardId must be set";
-            ReplicaResult result;
-            try (Releasable ignored = acquireReplicaOperationLock(request.shardId
    ↪ (), request.primaryTerm())) {
-                 result = shardOperationOnReplica(request);
-            }
-            result.respond(new ResponseListener());
+            acquireReplicaOperationLock(request.shardId(), request.primaryTerm(),
    ↪  this);
         }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.4 which introduced global checkpoints that identify common history across replica shards at a given time.

Code level perspective. The listing below shows changes which capture the `response` of the replication of indexed `result` to replica shards. The `response` contains local checkpoints of the replica shards. Then the `response` is passed to the replication response.

```
         protected void doRun() throws Exception {
             setPhase(task, "replica");
+            final ReplicaResponse response;
             assert request.shardId() != null : "request shardId must be set";
             ReplicaResult result;
-            try (Releasable ignored = acquireReplicaOperationLock(request.shardId
    ↪ (), request.primaryTerm())) {
+            try (ShardReference replica = getReplicaShardReference(request.
    ↪ shardId(), request.primaryTerm())) {
                 result = shardOperationOnReplica(request);
+                response = new ReplicaResponse(replica.routingEntry().
    ↪ allocationId().getId(), replica.getLocalCheckpoint());
             }
-            result.respond(new ResponseListener());
+            result.respond(new ResponseListener(response));
         }
```

B.2.10 Conflict 10

Category: Addition of statements in the Same area

Mainline Change

Category: Refactoring

Project level perspective. Internal engine's inner index and delete methods were refactored to extract common logic into single method. The refactoring was done to shrink the bytecode size of the inner index method so that the method can be

inlined, that is, optimization of method call done by just-in-time (JIT) compiler.

Code level perspective. The listing below shows changes which are part of common logic extraction from the internal engine's inner index and delete methods. The listing shows the extraction of the common logic for checking version conflicts from the inner delete method. In the listing, the condition `delete.versionType() .isVersionConflictForWrites(currentVersion, expectedVersion, deleted)` which check if there is version conflict for delete operation was extracted to a shared method `checkVersionConflict` that returns a boolean if there is conflict for the delete and index operations. For example, the condition `checkVersionConflict(delete, currentVersion, expectedVersion, deleted)` returns a boolean for the delete operation.

```
     private void innerDelete(Delete delete) throws IOException {
         try (Releasable ignored = acquireLock(delete.uid())) {
             ...
-            long updatedVersion;
-            long expectedVersion = delete.version();
-            if (delete.versionType().isVersionConflictForWrites(currentVersion,
    ↪ expectedVersion, deleted)) {
-                if (delete.origin().isRecovery()) {
-                    return;
-                } else {
-                    throw new VersionConflictEngineException(shardId, delete.type
    ↪ (), delete.id(),
-                            delete.versionType().explainConflictForWrites(
    ↪ currentVersion, expectedVersion, deleted));
-                }
-            }
-            updatedVersion = delete.versionType().updateVersion(currentVersion,
    ↪ expectedVersion);
-            final boolean found;
-            if (currentVersion == Versions.NOT_FOUND) {
-                // doc does not exist and no prior deletes
-                found = false;
-            } else if (versionValue != null && versionValue.delete()) {
-                // a "delete on delete", in this case, we still increment the
    ↪ version, log it, and return that version
-                found = false;
-            } else {
-                // we deleted a currently existing document
-                indexWriter.deleteDocuments(delete.uid());
-                found = true;
-            }
+            final long expectedVersion = delete.version();
+            if (checkVersionConflict(delete, currentVersion, expectedVersion,
    ↪ deleted)) return;
+
+            final long updatedVersion = updateVersion(delete, currentVersion,
    ↪ expectedVersion);
+
+            final boolean found = deleteIfFound(delete, currentVersion, deleted,
    ↪ versionValue);

             delete.updateVersion(updatedVersion, found);

-            if (delete.origin() != Operation.Origin.LOCAL_TRANSLOG_RECOVERY) {
-                final Translog.Location translogLocation = translog.add(new
    ↪ Translog.Delete(delete));
-                delete.setTranslogLocation(translogLocation);
-                versionMap.putUnderLock(delete.uid().bytes(), new
    ↪ DeleteVersionValue(updatedVersion, engineConfig.getThreadPool().
    ↪ estimatedTimeInMillis(), delete.getTranslogLocation()));
-            } else {
-                // we do not replay in to the translog, so there is no
```

```
-                // translog location; that is okay because real-time
-                // gets are not possible during recovery and we will
-                // flush when the recovery is complete
-                versionMap.putUnderLock(delete.uid().bytes(), new
    ↪ DeleteVersionValue(updatedVersion, engineConfig.getThreadPool().
    ↪ estimatedTimeInMillis(), null));
-            }
+            maybeAddToTranslog(delete, updatedVersion, Translog.Delete::new,
    ↪ DeleteVersionValue::new);
+        }
+    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as first change in B.2.3 which introduced local checkpoints.

Code level perspective. In the listing below, the condition `delete.origin() == Operation.Origin.PRIMARY` was introduced when updating the sequence number for the `delete` operation to check if the origin of the delete operation is the primary shard. The sequence number (see B.2.2) is the counter which is added to each write operation run on the shard to identify operations that are coming from the old failed primary shard. The last sequence number is used as the local checkpoint in the replica shards and then sent to the primary shards which update its global checkpoints (see B.2.4).

```
    private void innerDelete(Delete delete) throws IOException {
        try (Releasable ignored = acquireLock(delete.uid())) {
            ...

            long updatedVersion;
            long expectedVersion = delete.version();
            if (delete.versionType().isVersionConflictForWrites(currentVersion,
                ↪ expectedVersion, deleted)) {
                if (delete.origin().isRecovery()) {
                    return;
                } else {
                    throw new VersionConflictEngineException(shardId, delete.type
                        ↪ (), delete.id(),
-                                delete.versionType().explainConflictForWrites(
    ↪ currentVersion, expectedVersion, deleted));
+                                delete.versionType().explainConflictForWrites(
    ↪ currentVersion, expectedVersion, deleted));
                }
            }
            updatedVersion = delete.versionType().updateVersion(currentVersion,
                ↪ expectedVersion);
+
+           if (delete.origin() == Operation.Origin.PRIMARY) {
+               delete.updateSeqNo(seqNoService.generateSeqNo());
+           }
+
            final boolean found;
            ...
        }
    }
```

B.2.11 Conflict 11

Category: Modification and removal of statements

Mainline Change

Category: Test improvement

Project level perspective. Use of environment variables in replacing property placeholders (settings set when building xContent) was refactored so that tests can mock the behavior they need by obtaining environment variables without depending on external environment variables.

Code level perspective. Listing below shows a change to make environment variables visible to tests. A method `replacePropertyPlaceholders` was split to implement visibility of the environment variables to tests without depending on external environment variables. One method was made `public` and it delegates environment variables to the other method visible in the package. Tests can mock required behavior using the method visible in the package. In this way, tests do not rely on external environment variables.

```
+         public Builder replacePropertyPlaceholders() {
+             return replacePropertyPlaceholders(System::getenv);
          }
          ...
-         public Builder replacePropertyPlaceholders() {
+         // visible for testing
+         Builder replacePropertyPlaceholders(Function<String, String> getenv) {
              PropertyPlaceholder propertyPlaceholder = new PropertyPlaceholder("${
                  ↪ ", "}", false);
              PropertyPlaceholder.PlaceholderResolver placeholderResolver = new
                  ↪ PropertyPlaceholder.PlaceholderResolver() {
-                 @Override
-                 public String resolvePlaceholder(String placeholderName) {
-                     if (placeholderName.startsWith("env.")) {
-                         // explicit env var prefix
-                         return System.getenv(placeholderName.substring("env."
    ↪ .length()));
-                     }
-                     String value = System.getProperty(placeholderName);
-                     if (value != null) {
-                         return value;
-                     }
-                     value = System.getenv(placeholderName);
-                     if (value != null) {
-                         return value;
-                     }
-                     return map.get(placeholderName);
+                 @Override
+                 public String resolvePlaceholder(String placeholderName) {
+                     final String value = getenv.apply(placeholderName);
+                     if (value != null) {
+                         return value;
                      }
+                     return map.get(placeholderName);
+                 }
          ...
          }
```

Branch Change

Category: Refactoring

Project level perspective. Property placeholder (placeholder for property such as system property or environment variable) was moved to setting. The change was made so that the property placeholder can only be accessible through setting.

Code level perspective. Listing below shows changes which are part of moving property placeholder to setting package. The listing shows a method for replacing a property placeholder. The iterator variable `entryItr` holds all the settings for the xContent builder. Then for each element in the settings is replaced with specific setting set for the builder.

```
        public Builder replacePropertyPlaceholders() {
            PropertyPlaceholder propertyPlaceholder = new PropertyPlaceholder("${"
                ↪ , "}", false);
            PropertyPlaceholder.PlaceholderResolver placeholderResolver = new
                ↪ PropertyPlaceholder.PlaceholderResolver() {
                    @Override
                    public String resolvePlaceholder(String placeholderName) {
                        if (placeholderName.startsWith("env.")) {
                            // explicit env var prefix
                            return System.getenv(placeholderName.substring("env.".
                                ↪ length()));
                        }
            ...
                         return true;
                    }
                };
-           for (Map.Entry<String, String> entry : new HashMap<>(map).entrySet())
    ↪  {
-               String value = propertyPlaceholder.replacePlaceholders(entry.
    ↪ getKey(), entry.getValue(), placeholderResolver);
+           Iterator<Map.Entry<String, String>> entryItr = map.entrySet().
    ↪ iterator();
+           while (entryItr.hasNext()) {
+               Map.Entry<String, String> entry = entryItr.next();
+               if (entry.getValue() == null) {
+                   // a null value obviously can't be replaced
+                   continue;
+               }
+               String value = propertyPlaceholder.replacePlaceholders(entry.
    ↪ getValue(), placeholderResolver);
                // if the values exist and has length, we should maintain it in
                    ↪ the map
                // otherwise, the replace process resolved into removing it
                if (Strings.hasLength(value)) {
-                   map.put(entry.getKey(), value);
+                   entry.setValue(value);
                } else {
-                   map.remove(entry.getKey());
+                   entryItr.remove();
                }
            }
            return this;
        }
```

B.2.12 Conflict 12

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Document mapping (defining how document and its fields are indexed and stored) was refactored to remove duplicate or dead implementation of mapping meta-fields. The meta-fields of the document include _index (index which document belongs), _id (document ID), _uid (comprise document ID and mapping type), and _type (document's mapping type specified by user, for example product catalog might be stored in catalog type, which divide document into logical groups). Also, in this change, the _index field was made not configurable by the user.

Code level perspective. Listing below shows a change which is part of refactoring implementation of document's meta-fields in document mapping. A parameter `uidField` which is a document field that holds document ID and document's mapping type was removed in a method which parse document for indexing.

```
    private ParsedDocument testParsedDocument(String uid, String id, String type,
        ↪ String routing, long timestamp, long ttl, ParseContext.Document
        ↪ document, BytesReference source, Mapping mappingsUpdate) {
        Field uidField = new Field("_uid", uid, UidFieldMapper.Defaults.FIELD_TYPE
            ↪ );
        Field versionField = new NumericDocValuesField("_version", 0);
        document.add(uidField);
        document.add(versionField);
-       return new ParsedDocument(uidField, versionField, id, type, routing,
    ↪ timestamp, ttl, Arrays.asList(document), source, mappingsUpdate);
         ...
+       return new ParsedDocument(versionField, id, type, routing, timestamp, ttl,
    ↪  Arrays.asList(document), source, mappingsUpdate);
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2.

Code level perspective. In the listing below, a parameter `seqNoField` for document field which holds a counter to enforce semantic of primary terms. The counter (a sequence number) is passed when parsing a document during indexing.

```
     private ParsedDocument testParsedDocument(String uid, String id, String type,
         ↪  String routing, long timestamp, long ttl, ParseContext.Document
         ↪ document, BytesReference source, Mapping mappingsUpdate) {
         Field uidField = new Field("_uid", uid, UidFieldMapper.Defaults.
             ↪ FIELD_TYPE);
         Field versionField = new NumericDocValuesField("_version", 0);
+        Field seqNoField = new NumericDocValuesField("_seq_no", 0);
         document.add(uidField);
         document.add(versionField);
-        return new ParsedDocument(uidField, versionField, id, type, routing,
    ↪ timestamp, ttl, Arrays.asList(document), source, mappingsUpdate);
+        return new ParsedDocument(uidField, versionField, seqNoField, id, type,
    ↪ routing, timestamp, ttl, Arrays.asList(document), source, mappingsUpdate);
     }
```

B.2.13 Conflict 13

Category: Change of Method call or object creation

Mainline Change

Category: Breaking change fix

Project level perspective. Metric for available memory for all nodes in the cluster was removed in the cluster statistics. The metric was removed since it remained when statistics for specific operating system (OS) were removed and it has no use as it provides total available memory used in all nodes through the cluster. Therefore, statistics of available memory in all nodes in the cluster can be obtained from the individual node statistics.

Code level perspective. The listing below shows a change which was made to remove metric for available memory in all nodes in the cluster. The metric `availableMemory` was removed in the method `readFrom` which read stream `in` transferred to xContent.

```
    public static class OsStats implements ToXContent, Streamable {
        ...
        public void readFrom(StreamInput in) throws IOException {
            availableProcessors = in.readVInt();
            allocatedProcessors = in.readVInt();
-           availableMemory = in.readLong();
            int size = in.readVInt();
            names.clear();
            for (int i = 0; i < size; i++) {
                names.addTo(in.readString(), in.readVInt());
            }
        }
    }
```

Branch Change

Category: Feature enhancement

Project level perspective. Node setting which was used to instantiate a node based client was removed. The setting allows to set the node which is used as a client and route the client operations to other nodes that the operations need to execute on. The setting was removed as there are settings for making a node as a master node which among other responsibilities it tracks other nodes in the same cluster and assign shards to nodes, ingest node which processes documents before they are indexed, and data node which holds the indexed documents indexed in the shards. Therefore, the settings for the master and data nodes can replace the setting for the client.

Code level perspective. The listing below shows changes that were part of removing the setting which make the node based client. The class for cluster nodes statistics was changed to implement a `Writeable` instead of `Streamable`. Therefore,

with this change, the method `readFrom` which reads from the stream was changed to return an instance of operating system statistics.

```
     public static class OsStats implements ToXContent, Streamable {
         ...
-        public void readFrom(StreamInput in) throws IOException {
-            availableProcessors = in.readVInt();
-            allocatedProcessors = in.readVInt();
-            availableMemory = in.readLong();
-            int size = in.readVInt();
-            names.clear();
-            for (int i = 0; i < size; i++) {
-                names.addTo(in.readString(), in.readVInt());
-            }
+        public OsStats readFrom(StreamInput in) throws IOException {
+            return new OsStats(in);
         }
     }
```

B.2.14 Conflict 14

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. A new module for node connection management was introduced to separate cluster module which manages cluster state and transport module which is used for the nodes communication within the cluster. The node connection management module connects and disconnects nodes connection to the cluster when they are added and removed from the cluster respectively. The change was made to remove the dependency between the cluster and transport modules.

Code level perspective. Listing below shows changes that led to the merge conflict which are part of separating the dependency between cluster and transport modules. The method `buildAttributes` that builds attributes for node discovering and master node selection in the cluster was changed to `buildLocalNode` which returns an instance of the node in the cluster when the node transport address is passed.

```
-    public Map<String, String> buildAttributes() {
+    public DiscoveryNode buildLocalNode(TransportAddress publishAddress) {
         Map<String, String> attributes = new HashMap<>(settings.getByPrefix("node
             ↪ .").getAsMap());
         attributes.remove("name"); // name is extracted in other places
         if (attributes.containsKey("client")) {
             ...
         }
         if (attributes.containsKey("data")) {
             ...
         }

         for (CustomAttributesProvider provider : customAttributesProviders) {
             ...
         }

-        return attributes;
```

```
+        final String nodeId = generateNodeId(settings);
+        return new DiscoveryNode(settings.get("node.name"), nodeId,
    ↪ publishAddress, attributes, version);
     }
```

Branch Change

Category: Feature enhancement

Project level perspective. Same as B.2.13 which removed the setting used to set the node as a client. The client node was used to route operations to other nodes required to execute the operations. However, the data and master nodes can execute the operations that they are responsible with. Therefore, the role of the client node was unnecessary. Another change that was also part of removing the client node setting is introduction of concept of node roles when discovering nodes in the cluster.

Code level perspective. Listing below shows changes which removed client node setting and introduced the concept of roles into node discovery in the cluster. An exception `IllegalArgumentException` is thrown if the setting `node.client` for setting the node as the client node is passed when the nodes are configured. Furthermore, the node role `DATA` was added when building attributes for discovering the nodes in the cluster.

```
     public Map<String, String> buildAttributes() {
-        Map<String, String> attributes = new HashMap<>(settings.getByPrefix("node
    ↪ .").getAsMap());
+        Map<String, String> attributes = new HashMap<>(Node.NODE_ATTRIBUTES.get(
    ↪ this.settings).getAsMap());
         attributes.remove("name"); // name is extracted in other places
         if (attributes.containsKey("client")) {
-            if (attributes.get("client").equals("false")) {
-                attributes.remove("client"); // this is the default
-            } else {
-                // if we are client node, don't store data ...
-                attributes.put("data", "false");
-            }
+            throw new IllegalArgumentException("node.client setting is no longer
    ↪ supported, use " + Node.NODE_MASTER_SETTING.getKey()
+                + ", " + Node.NODE_DATA_SETTING.getKey() + " and " + Node.
    ↪ NODE_INGEST_SETTING.getKey() + " explicitly instead");
         }
-        if (attributes.containsKey("data")) {
-            if (attributes.get("data").equals("true")) {
-                attributes.remove("data");
+        //nocommit why don't we remove master as well if it's true? and ingest?
+        if (attributes.containsKey(DiscoveryNode.Role.DATA.getRoleName())) {
+            if (attributes.get(DiscoveryNode.Role.DATA.getRoleName()).equals("
    ↪ true")) {
+                attributes.remove(DiscoveryNode.Role.DATA.getRoleName());
             }
         }
         for (CustomAttributesProvider provider : customAttributesProviders) {
            ...
         }

         return attributes;
     }
```

B.2.15 Conflict 15

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. Resolution of the index name to index instances was modified so that to be performed early when there is possibility of cluster state changes. Before the change, the resolution was performed when there is a need for it. This was deemed as a problem specifically when there is a request to the index is carried out and at the same time there are index changes due mapping update sent by a node to the master node. When sending the mapping update to the master node, the index name is used to identify the index in the cluster. Therefore, with this change, to solve the problem, the mapping update uses a concrete index which is a tuple that containing index name and UUID (see also B.2.16). Therefore, the mapping update will be performed if the index matches the tuple.

Code level perspective. The listing below shows a change which was part of resolving index name to index instance early. The method `readFrom` from `IngestStats` class, which build statistics for document preprocessing before indexing, was referenced when reading a stream for ingest statistics in the node statistics.

```
    public void readFrom(StreamInput in) throws IOException {
        ...
        breaker = AllCircuitBreakerStats.readOptionalAllCircuitBreakerStats(in);
        scriptStats = in.readOptionalStreamable(ScriptStats::new);
        discoveryStats = in.readOptionalStreamable(() -> new DiscoveryStats(null)
            ↪ );
-       ingestStats = in.readOptionalWritable(IngestStats.PROTO);
+       ingestStats = in.readOptionalWritable(IngestStats.PROTO::readFrom);
    }
```

Branch Change

Category: Test improvement

Project level perspective. An infrastructure was added to run REST tests on a cluster with multiple nodes that use two different Elasticsearch minor versions. The infrastructure can be used for backward compatibility tests.

Code level perspective. The listing below shows a change which was part of adding the infrastructure for running REST tests on the cluster with nodes that have two different minor versions. The value passed when reading a stream for the ingest statistics in the node statistics was changes from static member `PROTO` which is an ingest prototype to constructor reference of the class `IngestStats` which is used for the document preprocessing statistics.

```
    public void readFrom(StreamInput in) throws IOException {
        ...
        breaker = AllCircuitBreakerStats.readOptionalAllCircuitBreakerStats(in);
```

```
            scriptStats = in.readOptionalStreamable(ScriptStats::new);
            discoveryStats = in.readOptionalStreamable(() -> new DiscoveryStats(null)
                ↪ );
-           ingestStats = in.readOptionalWritable(IngestStats.PROTO);
+           ingestStats = in.readOptionalWritable(IngestStats::new);
        }
```

B.2.16 Conflict 16

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. An index lookup in indices module, which provide services such as the index and mapping management, used in search and other requests was modified to use index UUID (Universally Unique Identifier) instead of index name. The change was made to look up for the index UUID in the index instance, which is a tuple containing UUID and name. This change prevents a situation of modifying a wrong index in case there are two indices with same name but different UUID.

Code level perspective. In the listing below, a parameter `context.shardTarget().index()` which pass index name when parsing search request content for search suggestions was removed. The index name was passed so that the search suggester can access the index it operates on. Therefore, this change aims to prevent modification of wrong index by not passing the index name when the search content is parsed for suggestions.

```
        public void parse(XContentParser parser, SearchContext context) throws
            ↪ Exception {
            SuggestionSearchContext suggestionSearchContext = parseInternal(parser,
                ↪ context.mapperService(), context.fieldData(),
-                   context.shardTarget().index(), context.shardTarget().shardId());
+               context.shardTarget().shardId());
            context.suggest(suggestionSearchContext);
        }
```

Branch Change

Category: Refactoring

Project level perspective. Suggestion context builder, which build context for suggestion when searching documents in an index, was added to phrase suggestion builder, which build correct phrases using word tokens for search suggestion. Also, search suggestion context builder was added to the top-level suggestion builder. The changes were made to add context builders for search suggestion in the top level. This change was part of refactoring suggest feature which suggests terms that are looking similar by using a suggester based on a text that is provided.

Code level perspective. In the listing below, the parameters `context .shardTarget() .index()` and `context.shardTarget() .shardId()`, which pass index name and shard ID respectively when parsing search content for search suggestion, were removed and replaced with the parameter `context.getQueryShardContext()` that get shard context from a context object which is used to create queries on shard level.

```
    public void parse(XContentParser parser, SearchContext context) throws
        ↪ Exception {
-       SuggestionSearchContext suggestionSearchContext = parseInternal(parser,
    ↪ context.mapperService(), context.fieldData(),
-               context.shardTarget().index(), context.shardTarget().shardId());
+       SuggestionSearchContext suggestionSearchContext = parseInternal(parser,
    ↪ context.getQueryShardContext());
        context.suggest(suggestionSearchContext);
    }
```

B.2.17 Conflict 17

Category: Addition of statements in the Same area

Mainline Change

Category: Feature enhancement

Project level perspective. The logic for parsing Azure storage settings was simplified. The change was made to take advantage of the new implemented settings infrastructure which made Elasticsearch module settings to be resettable and transactionally updateable. Therefore, if there is an error, the settings will not be updated.

Code level perspective. In the listing below, a setting field `key`, which before the change it was an instance variable of type `String`, was converted to string since it is the instance variable of type `Key` which sets a key for a setting.

```
    public final XContentBuilder toXContent(XContentBuilder builder, Params
        ↪ params) throws IOException {
        builder.startObject();
-       builder.field("key", key);
+       builder.field("key", key.toString());
        builder.field("type", scope.name());
        builder.field("dynamic", dynamic);
        builder.field("is_group_setting", isGroupSetting());
        ...
    }
```

Branch Change

Category: Feature enhancement

Project level perspective. Support for filtering settings that might contain information which are sensitive such as cloud platform keys was defined to allow to filter automatically the information when creating a setting module. Similar to the mainline change, this change was made after the new infrastructure of setting module

was implemented.

Code level perspective. The listing below shows changes which were made to support filtering of settings automatically. The setting `type` and `dynamic` fields, which specifies type of the setting based on the scope and if the setting can be updated while a cluster is running respectively, were removed and replaced with setting `properties` which specifies dynamism and scope of the setting. The list of the properties that can be set are filtered, dynamic, cluster scope, node scope, and index scope which specifies whether the setting can be filtered or not, if the setting can be updated while the cluster is running, if the setting is applicable to cluster level, if the setting is applicable to node level, and if the setting is applicable when indexing.

```
    public final XContentBuilder toXContent(XContentBuilder builder, Params
        ↪ params) throws IOException {
        builder.startObject();
        builder.field("key", key);
-       builder.field("type", scope.name());
-       builder.field("dynamic", dynamic);
+       builder.field("properties", properties);
        builder.field("is_group_setting", isGroupSetting());
        builder.field("default", defaultValue.apply(Settings.EMPTY));
        builder.endObject();
        ...
    }
```

B.2.18 Conflict 18

Category: Change of Method call or object creation

Mainline Change

Category: Framework removal

Project level perspective. Request and query caches for indices were moved to indices service provider which provides service for the operations on the indices such as removing an index and providing node statistics. The change was made to remove Guice from the caches.

Code level perspective. The listing is part of moving request and query caches to indices service after removing dependency to Guice framework. The parameter `indicesService.getIndicesQueryCache()` was added to get query cache for the common shards statistics in the node.

```
    protected ClusterStatsNodeResponse nodeOperation(ClusterStatsNodeRequest
        ↪ nodeRequest) {
        ...
        List<ShardStats> shardsStats = new ArrayList<>();
        for (IndexService indexService : indicesService) {
            for (IndexShard indexShard : indexService) {
                if (indexShard.routingEntry() != null && indexShard.routingEntry
                    ↪ ().active()) {
                    // only report on fully started shards
-                   shardsStats.add(new ShardStats(indexShard.routingEntry(),
    ↪ indexShard.shardPath(), new CommonStats(indexShard, SHARD_STATS_FLAGS),
    ↪ indexShard.commitStats()));
```

```
+                       shardsStats.add(new ShardStats(indexShard.routingEntry(),
    ↪ indexShard.shardPath(), new CommonStats(indicesService.getIndicesQueryCache
    ↪ (), indexShard, SHARD_STATS_FLAGS), indexShard.commitStats()));
                    }
                }
            }
            ...
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.3 which introduced local checkpoints.

Code level perspective. The listing below shows changes that led to the merge conflict when the local checkpoints were introduced. The parameter `indexShard.seqNoStats()` was added to get statistics of the sequence numbers when indexing the shard. The sequence numbers for individual shard are used as checkpoints which provide history across all shards in the node.

```
    protected ClusterStatsNodeResponse nodeOperation(ClusterStatsNodeRequest
        ↪ nodeRequest) {
        ...
        List<ShardStats> shardsStats = new ArrayList<>();
        for (IndexService indexService : indicesService) {
            for (IndexShard indexShard : indexService) {
                if (indexShard.routingEntry() != null && indexShard.routingEntry
                    ↪ ().active()) {
                    // only report on fully started shards
-                   shardsStats.add(new ShardStats(indexShard.routingEntry(),
    ↪ indexShard.shardPath(), new CommonStats(indexShard, SHARD_STATS_FLAGS),
    ↪ indexShard.commitStats()));
+                   shardsStats.add(new ShardStats(indexShard.routingEntry(),
    ↪ indexShard.shardPath(),
+                           new CommonStats(indexShard, SHARD_STATS_FLAGS),
    ↪ indexShard.commitStats(), indexShard.seqNoStats()));
                }
            }
        }
        ...
    }
```

B.2.19 Conflict 19

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. There are two changes from the common ancestor version that led to this conflict. In the first change, the UUID were made available in index properties, shard level components, and shard routing information. This change relates to B.2.16, whereUUID were used for the index lookup in indices services instead of index name. This change was a first step to use UUIDs instead of name

when comparing index, for example when looking up the index for modification. In the second change, version information that is stored in the shard state was removed in shard routing information. Instead of the version information, the allocation IDs (IDs generated when shard is allocated to a cluster) are used to allocate a primary shard, for example when a cluster is restarted. The change was made since there is no use case for the version information.

Code level perspective. Listing below shows two changes that led to the merge conflict. In the first change, a parameter, that is `"_na_"`, was passed to shard level components (`ShardId`) when setting a cluster state for testing. On the other hand, in the second change, a parameter that pass version information when creating a new shard routing information was removed. In the code, version `1` passed in the method `newShardRouting` was removed.

```
    void setClusterState(TestClusterService clusterService, String index) {
        ...
            for (int j = 0; j < numberOfShards; j++) {
-               final ShardId shardId = new ShardId(index, ++shardIndex);
-               ShardRouting shard = TestShardRouting.newShardRouting(index,
    ↪ shardId.getId(), node.id(), true, ShardRoutingState.STARTED, 1);
+               final ShardId shardId = new ShardId(index, "_na_", ++shardIndex);
+               ShardRouting shard = TestShardRouting.newShardRouting(index,
    ↪ shardId.getId(), node.id(), true, ShardRoutingState.STARTED);
                IndexShardRoutingTable.Builder indexShard = new
                    ↪ IndexShardRoutingTable.Builder(shardId);
                indexShard.addShard(shard);
                indexRoutingTable.addIndexShard(indexShard.build());
            }
        ...
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Primary terms were introduced to track how many times a primary shard is selected among replica shards after a previous primary shard failed. The primary terms were introduced to identify operations that come from the failed primary shard. This change relates to B.2.2 and it is a first step of sequence numbers introduction.

Code level perspective. Listing below shows changes made which introduce primary terms in a method which set cluster state for testing. A new parameter was added to a method `newShardRouting` which create new shard routing information. In the test, a primary term `primaryTerm` is passed to the shard routing information.

```
    void setClusterState(TestClusterService clusterService, String index) {
        int numberOfNodes = randomIntBetween(3, 5);
        DiscoveryNodes.Builder discoBuilder = DiscoveryNodes.builder();
        IndexRoutingTable.Builder indexRoutingTable = IndexRoutingTable.builder(
            ↪ index);
        ...
            for (int j = 0; j < numberOfShards; j++) {
                final ShardId shardId = new ShardId(index, ++shardIndex);
-               ShardRouting shard = TestShardRouting.newShardRouting(index,
    ↪ shardId.getId(), node.id(), true, ShardRoutingState.STARTED, 1);
```

```
+                 final int primaryTerm = randomInt(200);
+                 ShardRouting shard = TestShardRouting.newShardRouting(index,
    ↪ shardId.getId(), node.id(), primaryTerm, true, ShardRoutingState.STARTED,
    ↪ 1);
                  IndexShardRoutingTable.Builder indexShard = new
                      ↪ IndexShardRoutingTable.Builder(shardId);
                  indexShard.addShard(shard);
                  indexRoutingTable.addIndexShard(indexShard.build());
      ...
    }
```

B.2.20 Conflict 20

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. Same as the second change in B.2.19.

Code level perspective. Listing below shows changes to remove version information in shard routing information. Same as in B.2.19, the parameter that pass the version information to the method which create new shard routing information was removed. In the code, a test version texttt1 was removed from a test shard routing information textttTestShardRouting.

```
    public void testCanRemainWithShardRelocatingAway() {
        ...
-       ShardRouting firstRouting = TestShardRouting.newShardRouting("test", 0, "
    ↪ node1", null, null, true, ShardRoutingState.STARTED, 1);
-       ShardRouting secondRouting = TestShardRouting.newShardRouting("test", 1,
    ↪ "node1", null, null, true, ShardRoutingState.STARTED, 1);
+       ShardRouting firstRouting = TestShardRouting.newShardRouting("test", 0, "
    ↪ node1", null, null, true, ShardRoutingState.STARTED);
+       ShardRouting secondRouting = TestShardRouting.newShardRouting("test", 1,
    ↪ "node1", null, null, true, ShardRoutingState.STARTED);
        RoutingNode firstRoutingNode = new RoutingNode("node1", discoveryNode1,
            ↪ Arrays.asList(firstRouting, secondRouting));
        RoutingTable.Builder builder = RoutingTable.builder().add(
          ...
        );
        ClusterState clusterState = ClusterState.builder(baseClusterState).
            ↪ routingTable(builder.build()).build();
  ...
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.19.

Code level perspective. Same as B.2.19, a new parameter was added to method `newShardRouting` to pass primary terms when a new shard routing information is

created. In the listing, `null` primary term was passed when testing shard routing information were created.

```
    public void testCanRemainWithShardRelocatingAway() {
        ...
-       ShardRouting firstRouting = TestShardRouting.newShardRouting("test", 0, "
    ↪ node1", null, null, true, ShardRoutingState.STARTED, 1);
-       ShardRouting secondRouting = TestShardRouting.newShardRouting("test", 1,
    ↪ "node1", null, null, true, ShardRoutingState.STARTED, 1);
+       ShardRouting firstRouting = TestShardRouting.newShardRouting("test", 0, "
    ↪ node1", null, null, 1, true, ShardRoutingState.STARTED, 1);
+       ShardRouting secondRouting = TestShardRouting.newShardRouting("test", 1,
    ↪ "node1", null, null, 1, true, ShardRoutingState.STARTED, 1);
        RoutingNode firstRoutingNode = new RoutingNode("node1", discoveryNode1,
            ↪ Arrays.asList(firstRouting, secondRouting));
        RoutingTable.Builder builder = RoutingTable.builder().add(
                IndexRoutingTable.builder("test")
                        .addIndexShard(new IndexShardRoutingTable.Builder(new
                            ↪ ShardId("test", 0))
                                .addShard(firstRouting)
                                .build()
                        )
                        .addIndexShard(new IndexShardRoutingTable.Builder(new
                            ↪ ShardId("test", 1))
                                .addShard(secondRouting)
                                .build()
                        )
        );
        ...
    }
```

B.2.21 Conflict 21

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. Same as the first change in B.2.19 which added UUID stored in index setting available to shard level components, index properties, and shard routing information.

Code level perspective. Listing below shows a change which was part of making index UUID available to shard routing information, shard components, and index properties. The value for shard ID passed on method `newUnwasassigned`, which create new unassigned shard during shard routing, changed from `"test"` to `indexMetaData.getIndex()` which retrieve shard ID from the index metadata.

```
     public void testCanRemainUsesLeastAvailableSpace() {
         ...
         shardRoutingMap.put(test_2, "/node1/most");

-        ShardRouting test_3 = ShardRouting.newUnassigned("test", 3, null, true,
    ↪ new UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "foo"));
+        ShardRouting test_3 = ShardRouting.newUnassigned(indexMetaData.getIndex()
    ↪ , 3, null, true, new UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "
    ↪ foo"));
         ShardRoutingHelper.initialize(test_3, node_1.getId());
         ShardRoutingHelper.moveToStarted(test_3);
         ...
```

```
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.19 which introduced primary terms that track number of promotion of replica shard to primary shard.

Code level perspective. Listing belows shows a change which introduce primary terms. A parameter was added to a method `newUnwasassigned` that pass primary term. In the change, the priary term `1` was passes for testing creating new unassigned shard during shard routing.

```
    public void testCanRemainUsesLeastAvailableSpace() {
        ...
        assertEquals(0l, DiskThresholdDecider.sizeOfRelocatingShards(node, info,
            ↪ true, "/dev/some/other/dev"));

-       ShardRouting test_3 = ShardRouting.newUnassigned("test", 3, null, false,
    ↪ new UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "foo"));
+       ShardRouting test_3 = ShardRouting.newUnassigned("test", 3, null, 1,
    ↪ false, new UnassignedInfo(UnassignedInfo.Reason.INDEX_CREATED, "foo"));
        ShardRoutingHelper.initialize(test_3, "node1");
        ...
    }
```

B.2.22 Conflict 22

Category: Change of Method call or object creation

Mainline Change

Category: Bug fix

Project level perspective. A performance bug in filter or filters aggregation was fixed. The filter aggregation is calculation of metrics using document fields by grouping documents into single bucket using a specified criterion based on the aggregation type. On the other hand, the filters aggregation groups the documents into multiple buckets and each bucket is associated to documents that match a filter (the criteria which was specified). The bug was due to creation of query weights each time when filter aggregation is below terms aggregation (metrics calculated based of a multi-bucket value source) with cardinality (number of term occurrence in documents) of 1000. For example, when searching documents using exact term specified in the inverted index (Lucene index), the term will be searched in every segment (sub-index). To fix this bug, a query weight is created once and iterators are used to look through in all segments.

Code level perspective. Listing below shows changes which are part of fixing the performance bug in the filters aggregation. The iterator was added to collect filters `filters` if the `searcher`, which search provided index, is not equal to the

context of index search `contextSearcher`. Then, the filters are passed to filter aggregators `FiltersAggregator`, which group the documents that match the filters for aggregration.

```
        public Aggregator createInternal(AggregationContext context, Aggregator
            ↪ parent, boolean collectsFromSingleBucket,
                List<PipelineAggregator> pipelineAggregators, Map<String, Object>
                    ↪  metaData) throws IOException {
-           return new FiltersAggregator(name, factories, filters, keyed,
    ↪ otherBucketKey, context, parent, pipelineAggregators, metaData);
+           IndexSearcher contextSearcher = context.searchContext().searcher();
+           if (searcher != contextSearcher) {
+               searcher = contextSearcher;
+               weights = new Weight[filters.size()];
+               for (int i = 0; i < filters.size(); ++i) {
+                   KeyedFilter keyedFilter = filters.get(i);
+                   this.weights[i] = contextSearcher.createNormalizedWeight(
    ↪ keyedFilter.filter, false);
+               }
+           }
+           return new FiltersAggregator(name, factories, keys, weights, keyed,
    ↪ otherBucketKey, context, parent, pipelineAggregators, metaData);
        }
    }
```

Branch Change

Category: Refactoring

Project level perspective. Filters Aggregation were refactored to allow them to be passed to the coordinating node and serialized to the shards. The coordinating node receives a request such as search request from a client, forwards the request to the nodes that hold the data, receives results from the data node, and creates a set of results which is returned to the client.

Code level perspective. As part of serializing filters aggregation, a parameter `otherBucketKey` was replaced with `otherBucket ? otherBucketKey : null` to check if an option to include other bucket which contains document that do not match the given filters was set. The parameter `otherBucket` is used in case a client wants to include aggregation of other buckets in a search response.

```
        public Aggregator createInternal(AggregationContext context, Aggregator
            ↪ parent, boolean collectsFromSingleBucket,
                List<PipelineAggregator> pipelineAggregators, Map<String, Object>
                    ↪  metaData) throws IOException {
-           return new FiltersAggregator(name, factories, filters, keyed,
    ↪ otherBucketKey, context, parent, pipelineAggregators, metaData);
+           return new FiltersAggregator(name, factories, filters, keyed,
    ↪ otherBucket ? otherBucketKey : null, context, parent,
+                   pipelineAggregators, metaData);
+       }
```

B.2.23 Conflict 23

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. The allocation IDs which are used when selecting a primary shard and generated when shard is allocated to a cluster were added to transport action which fetches the shards version in the nodes during the primary shard allocation. The allocation IDs were added to the transport action to be used when allocating the primary shard during cluster or node restart. When the node or cluster restarted, the transport action is used to identify which shards were primary shard and hold recent shard versions. Therefore, the allocation IDS were added to recover the right active shards.

Code level perspective. The listing below shows change which were part of adding the allocation IDs to the transport action. An expression `TestShardRouting.newShardRouting( shardId.getIndex(), shardId.getId(), node1.id(), true, ShardRoutingState.STARTED, 10)` which pass a primary shard when building routing table information was extracted to a variable `primaryShard`. Furthermore, a method `putActiveAllocationIds` was added to add active shard allocation IDs when building a metadata `metaData` during the node recovery.

```
     private RoutingAllocation onePrimaryOnNode1And1ReplicaRecovering(
         ↪ AllocationDeciders deciders) {
+        ShardRouting primaryShard = TestShardRouting.newShardRouting(shardId.
    ↪ getIndex(), shardId.getId(), node1.id(), true, ShardRoutingState.STARTED,
    ↪ 10);
         MetaData metaData = MetaData.builder()
-                .put(IndexMetaData.builder(shardId.getIndex()).settings(settings(
    ↪ Version.CURRENT)).numberOfShards(1).numberOfReplicas(0))
+                .put(IndexMetaData.builder(shardId.getIndex()).settings(settings(
    ↪ Version.CURRENT))
+                    .numberOfShards(1).numberOfReplicas(1)
+                    .putActiveAllocationIds(0, new HashSet<>(Arrays.asList(
    ↪ primaryShard.allocationId().getId()))))
                 .build();
         RoutingTable routingTable = RoutingTable.builder()
                 .add(IndexRoutingTable.builder(shardId.getIndex())
                                 .addIndexShard(new IndexShardRoutingTable.Builder
                                     ↪ (shardId)
-                                        .addShard(TestShardRouting.
    ↪ newShardRouting(shardId.getIndex(), shardId.getId(), node1.id(), true,
    ↪ ShardRoutingState.STARTED, 10))
+                                        .addShard(primaryShard)
                                         .addShard(TestShardRouting.
                                             ↪ newShardRouting(shardId.getIndex()
                                             ↪ , shardId.getId(), node2.id(),
                                             ↪ null, null, false,
                                             ↪ ShardRoutingState.INITIALIZING,
                                             ↪ 10, new UnassignedInfo(
                                             ↪ UnassignedInfo.Reason.
                                             ↪ CLUSTER_RECOVERED, null)))
                                         .build())
                 )
                 .build();
         ...
     }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.19. The change introduced primary terms that track promotion of the replica shards to primary shards.

Code level perspective. Listing belows shows changes which introduce primary terms in a test of recovering a node with one primary shard and one replica shard. A parameter was added in method `newShardRouting` and `newUnassigned` to pass primary terms. In the listing, the primary term `1` was passed when creating new shard routing information and new unassigned shard.

```
    private RoutingAllocation onePrimaryOnNode1And1ReplicaRecovering(
        ↪ AllocationDeciders deciders) {
        ...
        RoutingTable routingTable = RoutingTable.builder()
                .add(IndexRoutingTable.builder(shardId.getIndex())
                                .addIndexShard(new IndexShardRoutingTable.Builder
                                    ↪ (shardId)
-                                       .addShard(TestShardRouting.
    ↪ newShardRouting(shardId.getIndex(), shardId.getId(), node1.id(), true,
    ↪ ShardRoutingState.STARTED, 10))
-                                       .addShard(ShardRouting.newUnassigned(
    ↪ shardId.getIndex(), shardId.getId(), null, false, new UnassignedInfo(reason
    ↪ , null)))
+                                       .addShard(TestShardRouting.
    ↪ newShardRouting(shardId.getIndex(), shardId.getId(), node1.id(), 1, true,
    ↪ ShardRoutingState.STARTED, 10))
+                                       .addShard(ShardRouting.newUnassigned(
    ↪ shardId.getIndex(), shardId.getId(), null, 1, false, new UnassignedInfo(
    ↪ reason, null)))
                                        .build())
                )
                .build();
        ...
    }
```

B.2.24 Conflict 24

Category: Change of Method call or object creation

Mainline Change

Category: Test improvement

Project level perspective. Test for shard inactiveness was improved so that not to fail when a document is being indexed while the time set for the shard inactiveness elapsed. The API that check if the shard is active was separated to have another that is used only for testing so that the test should not fail if the shard inactiveness time elapsed.

Code level perspective. Listing below shows changes made to separate a method `checkIdle` which check if the shard is idle when indexing a document. The method for testing purpose was made private and a parameter which pass inactive time set is passed.

```
-    public boolean checkIdle(long inactiveTimeNS) {
-        if (System.nanoTime() - lastWriteNS >= inactiveTimeNS) {
     ...
+    public boolean checkIdle() {
+        return checkIdle(inactiveTime.nanos());
+    }
+
+    final boolean checkIdle(long inactiveTimeNS) { // pkg private for testing
+        Engine engineOrNull = getEngineOrNull();
+        if (engineOrNull != null && System.nanoTime() - engineOrNull.
    ↪ getLastWriteNanos() >= inactiveTimeNS) {
             boolean wasActive = active.getAndSet(false);
             if (wasActive) {
                 updateBufferSize(IndexingMemoryController.
                     ↪ INACTIVE_SHARD_INDEXING_BUFFER, IndexingMemoryController.
                     ↪ INACTIVE_SHARD_TRANSLOG_BUFFER);
-                logger.debug("shard is now inactive");
-                indicesLifecycle.onShardInactive(this);
+                logger.debug("marking shard as inactive (inactive_time=[{}])
    ↪ indexing wise", inactiveTime);
+                indexEventListener.onShardInactive(this);
             }
         }

         return active.get() == false;
     }
```

Branch Change

Category: Feature enhancement

Project level perspective. Indexing buffer size for shards that have heavy indexing was optimized. The size of the buffer in the indexing process is controlled by the indexing buffer. The change was made to improve optimization of RAM usage for indexing since a default index buffer size of 10% of node heap is divided up and assigned equally to all shards that are active. The assignment of the node heap to the active shards does not consider shard usage. With this change, each shard is assigned unlimited indexing buffer and the most shard or shards that consuming most of the heap are asked to clear up the heap if total bytes used across all shards exceed the node heap.

Code level perspective. As part of implementing the logic for optimizing the indexing buffer, a method `checkIdle` which check if the shard is idle was made `void` and not to update buffer size if it is active.

```
-    public boolean checkIdle(long inactiveTimeNS) {
     ...
+    public void checkIdle(long inactiveTimeNS) {
         if (System.nanoTime() - lastWriteNS >= inactiveTimeNS) {
             boolean wasActive = active.getAndSet(false);
             if (wasActive) {
-                updateBufferSize(IndexingMemoryController.
    ↪ INACTIVE_SHARD_INDEXING_BUFFER, IndexingMemoryController.
    ↪ INACTIVE_SHARD_TRANSLOG_BUFFER);
                 logger.debug("shard is now inactive");
                 indicesLifecycle.onShardInactive(this);
             }
         }
-
-        return active.get() == false;
```

```
    }
```

B.2.25 Conflict 25

Category: Addition of statements in the Same area

Mainline Change

Category: Bug fix

Project level perspective. An indexing memory controller which track indexing status of shards was modified to not maintain individual status of each shard. With this change, the indexing memory controller checks status of all shards and resizes the indexing buffers based on shard status, that is, if the shard is idle or not. The change was made to fix a performance regression caused by scenario which version maps of two shards are never resized to defaults when an index is deleted and created and therefore the new shard is not detected by the controller since it has a same shard ID as the deleted shard. The scenario results to increase number of the index merges which affect the performance.

Code level perspective. Listing below shows changes made in controller test which are part of fixing performance regression issue. The shard status is obtained when indexing a shard instead of tracking individual shard. Therefore, a parameter of type `IndexShard`, which track indexing of the shard, is passed instead of `ShardId`, which injects shard ID to shard level components.

```
-       public void simulateIndexing(ShardId shardId) {
-           lastIndexTimeNanos.put(shardId, currentTimeInNanos());
-           if (indexingBuffers.containsKey(shardId) == false) {
+       public void simulateIndexing(IndexShard shard) {
+           lastIndexTimeNanos.put(shard, currentTimeInNanos());
+           if (indexingBuffers.containsKey(shard) == false) {
                // First time we are seeing this shard; start it off with
                    ↪ inactive buffers as IndexShard does:
-               indexingBuffers.put(shardId, IndexingMemoryController.
    ↪ INACTIVE_SHARD_INDEXING_BUFFER);
-               translogBuffers.put(shardId, IndexingMemoryController.
    ↪ INACTIVE_SHARD_TRANSLOG_BUFFER);
+               indexingBuffers.put(shard, IndexingMemoryController.
    ↪ INACTIVE_SHARD_INDEXING_BUFFER);
+               translogBuffers.put(shard, IndexingMemoryController.
    ↪ INACTIVE_SHARD_TRANSLOG_BUFFER);
            }
-           activeShards.add(shardId);
+           activeShards.add(shard);
            forceCheck();
        }
    }
```

Branch Change

Category: Bug fix

Project level perspective. An indexing buffer size for inactive shard was increased once they become active again. If the shard become active, it took up to 30 seconds for the indexing memory controller, which track indexing status of the shards, to increase the buffer size from the idle size to the one assigned along with other active shards. This delay could cause many index segments not written. To fix this issue, the indexing buffer is divided up to all active shards immediately once the inactive shard becomes active.

Code level perspective. Listing below shows changes which are part of fixing assignment of indexing buffer once the inactive shard becomes active. In the test `simulateIndexing` which simulate tracking of shard status when indexing, a variable `bytes` track buffer size of the shard during indexing. The tracking time `lastIndexTimeNanos` for shard indexing was removed since shard buffer size is re-assigned once the shard become active again.

```
         public void simulateIndexing(ShardId shardId) {
-            lastIndexTimeNanos.put(shardId, currentTimeInNanos());
-            if (indexingBuffers.containsKey(shardId) == false) {
-                // First time we are seeing this shard; start it off with
    ↪ inactive buffers as IndexShard does:
-                indexingBuffers.put(shardId, IndexingMemoryController.
    ↪ INACTIVE_SHARD_INDEXING_BUFFER);
-                translogBuffers.put(shardId, IndexingMemoryController.
    ↪ INACTIVE_SHARD_TRANSLOG_BUFFER);
+            Long bytes = indexBufferRAMBytesUsed.get(shardId);
+            if (bytes == null) {
+                bytes = 0L;
             }
-            activeShards.add(shardId);
+            // Each doc we index takes up a megabyte!
+            bytes += 1024*1024;
+            indexBufferRAMBytesUsed.put(shardId, bytes);
             forceCheck();
         }
     }
```

B.2.26 Conflict 26

Category: Addition of statements in the Same area

Mainline Change

Category: Refactoring

Project level perspective. The shared logic such as creating xContent response of a write operation (index, update and delete) response on a single document was consolidated to have a base logic. The write operation response is used to get back information about the execution of the operation and then contents such as xContent is created from the operation response. With this change, the responses are retrieved using shard IDs which the operations are executed on.

Code level perspective. Listing below shows added changes which were part of consolidating the shared logic for creating response from the write operation

response on the single document. A method `status` was added which returns a REST status of the delete operation.

```
+    public RestStatus status() {
+        if (found == false) {
+            return RestStatus.NOT_FOUND;
+        }
+        return super.status();
+    }
```

Branch Change

Category: Refactoring

Project level perspective. The responses from `put` and `delete` pipeline APIs, which are used for adding or updating and deleting preprocessors that preprocess documents before indexing, were made consistent with the delete and index APIs, which are used for deleting and adding or updating documents in a specific index respectively.

Code level perspective. Listing below shows changes added to make the put and delete pipeline APIs responses consistent with the delete and index APIs responses. A method `status` which returns status of the delete operation was added.

```
+    public RestStatus status() {
+        RestStatus status = getShardInfo().status();
+        if (isFound() == false) {
+            status = NOT_FOUND;
+        }
+        return status;
+    }
```

B.2.27 Conflict 27

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. The operation logic for requests that are executed first on the primary shard and followed on the replica shards was separated to have logic for routing and retrying failed operations, and logic for the delegation of the replication operations. The operation logic was refactored when performing replication of the operations from the primary shard to the replica shards.

Code level perspective. The listing below shows changes which are part of separating the operation logic when performing replication action. The shard ID (or shard version) passed on `getIndexShardOperationsCounter` was changed from `request.internalShardId` to `request.shardId()` which returns the shard ID where the operations should be run.

```
    private final class AsyncReplicaAction extends AbstractRunnable {
         protected void doRun() throws Exception {
-            try (Releasable shardReference = getIndexShardOperationsCounter(
    ↪ request.internalShardId)) {
-                shardOperationOnReplica(request.internalShardId, request);
+            assert request.shardId() != null : "request shardId must be set";
+            try (Releasable ignored = getIndexShardOperationsCounter(request.
    ↪ shardId())) {
+                shardOperationOnReplica(request);
+                if (logger.isTraceEnabled()) {
+                    logger.trace("action [{}] completed on shard [{}] for request
    ↪  [{}]", transportReplicaAction, request.shardId(), request);
+                }
             }
             channel.sendResponse(TransportResponse.Empty.INSTANCE);
         }
     }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2 which added the counter to identify write operations from the failed primary shard and enforced the primary terms logic (see B.2.19).

Code level perspective. The listing below shows a change which enforced the primary terms logic. A parameter which pass the primary term, `request.primaryTerm`, was added to the method `getIndexShardOperationsCounter` that gets number of operations during shard indexing.

```
    private final class AsyncReplicaAction extends AbstractRunnable {
         protected void doRun() throws Exception {
-            try (Releasable shardReference = getIndexShardOperationsCounter(
    ↪ request.internalShardId)) {
+            try (Releasable shardReference = getIndexShardOperationsCounter(
    ↪ request.internalShardId, request.primaryTerm)) {
                 shardOperationOnReplica(request.internalShardId, request);
             }
             channel.sendResponse(TransportResponse.Empty.INSTANCE);
         }
     }
```

B.2.28 Conflict 28

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Same as B.2.26 which refactored shared logic such as an xContent response of write operations on the single document.

Code level perspective. The listing below shows two same changes which are part of the consolidation of the shared logic of write operations. The changes were made to pass shard ID `shardId` instead of the document index `getResult.getIndex()` when creating new response for an operation on the document that did not interpreted as the write operation, that is, the operation did not update the document.

```
    protected Result prepare(UpdateRequest request, final GetResult getResult) {
        ...
        if (operation == null || "index".equals(operation)) {
            ...
        } else if ("delete".equals(operation)) {
            ...
            return new Result(deleteRequest, Operation.DELETE, updatedSourceAsMap
                ↪ , updateSourceContentType);
        } else if ("none".equals(operation)) {
-           UpdateResponse update = new UpdateResponse(getResult.getIndex(),
    ↪ getResult.getType(), getResult.getId(), getResult.getVersion(), false);
+           UpdateResponse update = new UpdateResponse(shardId, getResult.getType
    ↪ (), getResult.getId(), getResult.getVersion(), false);
            update.setGetResult(extractGetResult(request, request.index(),
                ↪ getResult.getVersion(), updatedSourceAsMap,
                ↪ updateSourceContentType, getResult.internalSourceRef()));
            return new Result(update, Operation.NONE, updatedSourceAsMap,
                ↪ updateSourceContentType);
        } else {
            logger.warn("Used update operation [{}] for script [{}], doing
                ↪ nothing...", operation, request.script.getScript());
-           UpdateResponse update = new UpdateResponse(getResult.getIndex(),
    ↪ getResult.getType(), getResult.getId(), getResult.getVersion(), false);
+           UpdateResponse update = new UpdateResponse(shardId, getResult.getType
    ↪ (), getResult.getId(), getResult.getVersion(), false);
            return new Result(update, Operation.NONE, updatedSourceAsMap,
                ↪ updateSourceContentType);
        }
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2 which added the counter to identify write operations from the failed primary shard. Also, it is similar to the mainline change, however, it was combined in broad change of adding the counters.

Code level perspective. The listing below shows changes which were made as part of introducing the counter for identifying operations from the failed primary shard. A parameter `new ShardId(getResult.getIndex(), request.shardId())` which pass a new information on a shard that the operation is executed on is passed when creating a new document update response, which for this case is the update response for a non-write operation.

```
    protected Result prepare(UpdateRequest request, final GetResult getResult) {
        ...
        if (operation == null || "index".equals(operation)) {
            ...
        } else if ("delete".equals(operation)) {
            ...
            return new Result(deleteRequest, Operation.DELETE, updatedSourceAsMap
                ↪ , updateSourceContentType);
```

```
         } else if ("none".equals(operation)) {
-            UpdateResponse update = new UpdateResponse(getResult.getIndex(),
    ↪ getResult.getType(), getResult.getId(), getResult.getVersion(), false);
+            UpdateResponse update = new UpdateResponse(new ShardId(getResult.
    ↪ getIndex(), request.shardId()), getResult.getType(), getResult.getId(),
    ↪ getResult.getVersion(), false);
             update.setGetResult(extractGetResult(request, request.index(),
                 ↪ getResult.getVersion(), updatedSourceAsMap,
                 ↪ updateSourceContentType, getResult.internalSourceRef()));
             return new Result(update, Operation.NONE, updatedSourceAsMap,
                 ↪ updateSourceContentType);
         } else {
             logger.warn("Used update operation [{}] for script [{}], doing
                 ↪ nothing...", operation, request.script.getScript());
-            UpdateResponse update = new UpdateResponse(getResult.getIndex(),
    ↪ getResult.getType(), getResult.getId(), getResult.getVersion(), false);
+            UpdateResponse update = new UpdateResponse(new ShardId(getResult.
    ↪ getIndex(), request.shardId()), getResult.getType(), getResult.getId(),
    ↪ getResult.getVersion(), false);
             return new Result(update, Operation.NONE, updatedSourceAsMap,
                 ↪ updateSourceContentType);
         }
     }
```

B.2.29 Conflict 29

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Same as B.2.26 which refactored shared logic such as an xContent response of write operations on the single document.

Code level perspective. The listing below shows changes made to unify the xContent response logic of the write operations. The changes were made so that the document update by a client `client` alerts `RestStatusToXContentListener`, which builds the xContent response, instead of `RestBuilderListener`, which builds a response based on the xContent builder.

```
     public void handleRequest(final RestRequest request, final RestChannel
         ↪ channel, final Client client) throws Exception {
-        client.update(updateRequest, new RestBuilderListener<UpdateResponse>(
    ↪ channel) {
-            @Override
-            public RestResponse buildResponse(UpdateResponse response,
    ↪ XContentBuilder builder) throws Exception {
-                builder.startObject();
-                ActionWriteResponse.ShardInfo shardInfo = response.getShardInfo()
    ↪ ;
-                builder.field(Fields._INDEX, response.getIndex())
-                        .field(Fields._TYPE, response.getType())
-                        .field(Fields._ID, response.getId())
-                        .field(Fields._VERSION, response.getVersion());
-
-                shardInfo.toXContent(builder, request);
-                if (response.getGetResult() != null) {
-                    builder.startObject(Fields.GET);
-                    response.getGetResult().toXContentEmbedded(builder, request);
-                    builder.endObject();
```

```
-                 }
-
-                 builder.endObject();
-                 RestStatus status = shardInfo.status();
-                 if (response.isCreated()) {
-                     status = CREATED;
-                 }
-                 return new BytesRestResponse(status, builder);
-             }
-         });
-     }
-
-     static final class Fields {
-         static final XContentBuilderString _INDEX = new XContentBuilderString("
    ↪ _index");
-         static final XContentBuilderString _TYPE = new XContentBuilderString("
    ↪ _type");
-         static final XContentBuilderString _ID = new XContentBuilderString("_id")
    ↪ ;
-         static final XContentBuilderString _VERSION = new XContentBuilderString("
    ↪ _version");
-         static final XContentBuilderString GET = new XContentBuilderString("get")
    ↪ ;
+         client.update(updateRequest, new RestStatusToXContentListener<>(channel))
    ↪ ;
     }
 }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.2.

Code level perspective. The listing below shows changes as part of adding the counter to identify the write operations from old primary shard. The changes were made to use a method `toXContent` from a new response base class `DocWriteResponse` created from the extracted shared logic of write operations on a single document and receive REST status `status` from the shard information in the operation response.

```
    public void handleRequest(final RestRequest request, final RestChannel
        ↪ channel, final Client client) throws Exception {
        client.update(updateRequest, new RestBuilderListener<UpdateResponse>(
            ↪ channel) {
            @Override
            public RestResponse buildResponse(UpdateResponse response,
                ↪ XContentBuilder builder) throws Exception {
                builder.startObject();
-               ActionWriteResponse.ShardInfo shardInfo = response.getShardInfo()
    ↪ ;
-               builder.field(Fields._INDEX, response.getIndex())
-                       .field(Fields._TYPE, response.getType())
-                       .field(Fields._ID, response.getId())
-                       .field(Fields._VERSION, response.getVersion());
-
-               shardInfo.toXContent(builder, request);
-               if (response.getGetResult() != null) {
-                   builder.startObject(Fields.GET);
-                   response.getGetResult().toXContentEmbedded(builder, request);
-                   builder.endObject();
-               }
-
```

```
+                   response.toXContent(builder, request);
                    builder.endObject();
-                   RestStatus status = shardInfo.status();
+                   RestStatus status = response.getShardInfo().status();
                    if (response.isCreated()) {
                        status = CREATED;
                    }
                    return new BytesRestResponse(status, builder);
                }
            });
    }
```

B.2.30 Conflict 30

Category: Addition of statements in the Same area

Mainline Change

Category: Feature enhancement

Project level perspective. The allocation IDs, which are generated during shard allocation to a cluster, were persisted to index metadata so that to be used when choosing a next primary shard. This change is part of an enhancement to allocate a primary shard based on the allocation IDs of active shards in a current cluster state, which provide state information of the cluster, when the cluster is restarted.

Code level perspective. The listing below shows changes added to persist the allocations IDs `allocationId` in the cluster state. The cluster state information can be filtered using five metrics which are; version, master_node, nodes, routing_table, metadata and blocks. The changes were added for the metadata metric which returns only the metadata of the response.

```
    public XContentBuilder toXContent(XContentBuilder builder, Params params)
        ↪ throws IOException {
        EnumSet<Metric> metrics = Metric.parseString(params.param("metric", "_all
            ↪ "), true);
        ...
        if (metrics.contains(Metric.METADATA)) {
            ...
            for (IndexMetaData indexMetaData : metaData()) {
                ...
+               builder.startObject(IndexMetaData.KEY_ACTIVE_ALLOCATIONS);
+               for (IntObjectCursor<Set<String>> cursor : indexMetaData.
    ↪ getActiveAllocationIds()) {
+                   builder.startArray(String.valueOf(cursor.key));
+                   for (String allocationId : cursor.value) {
+                       builder.value(allocationId);
+                   }
+                   builder.endArray();
+               }
+               builder.endObject();
+
                    builder.endObject();
                }
                builder.endObject();
                ...
        }
        ...
        return builder;
    }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as B.2.19 which introduced primary terms to track how many times a replica shard is promoted to a primary shard when an older primary shard failed so that to identify if there are operations are coming from the old primary shard.

Code level perspective. Listing belows shows changes which introduce primary terms when building response for the cluster state request. The changes were added when the request is filtered using metadata metric which returns only the metadata of the cluster state.

```
    public XContentBuilder toXContent(XContentBuilder builder, Params params)
        ↪ throws IOException {
        EnumSet<Metric> metrics = Metric.parseString(params.param("metric", "_all
            ↪ "), true);
        ...
        if (metrics.contains(Metric.METADATA)) {
            ...
            for (IndexMetaData indexMetaData : metaData()) {
                ...
+               builder.startObject("primary_terms");
+               for (int shard = 0; shard < indexMetaData.getNumberOfShards();
    ↪ shard++) {
+                   builder.field(Integer.toString(shard), indexMetaData.
    ↪ primaryTerm(shard));
+               }
+               builder.endObject();
+
                builder.endObject();
            }
            builder.endObject();
            ...
        }
        ...
        return builder;
    }
```

B.2.31 Conflict 31

Category: Addition of statements in the Same area

Mainline Change

Category: Feature enhancement

Project level perspective. Same as B.2.30.

Code level perspective. Listing below shows changes which are part of persisting allocation IDs to enhance allocation of primary during cluster restart. A variable `activeAllocationIds` was added to persist allocation IDs during creation of a new cluster's state *diff*. Furthermore, a new parameter was added to method `diff`, which create a *diff* of two immutable maps, in `DiffableUtils`. Therefore, the value for variables mappings, aliases, and customs were modified to pass the value

`DiffableUtils.getStringKeySerializer()` for the new parameter which returns serializer for the strings keys.

```
        public IndexMetaDataDiff(IndexMetaData before, IndexMetaData after) {
            ...
            settings = after.settings;
-           mappings = DiffableUtils.diff(before.mappings, after.mappings);
-           aliases = DiffableUtils.diff(before.aliases, after.aliases);
-           customs = DiffableUtils.diff(before.customs, after.customs);
+           mappings = DiffableUtils.diff(before.mappings, after.mappings,
    ↪ DiffableUtils.getStringKeySerializer());
+           aliases = DiffableUtils.diff(before.aliases, after.aliases,
    ↪ DiffableUtils.getStringKeySerializer());
+           customs = DiffableUtils.diff(before.customs, after.customs,
    ↪ DiffableUtils.getStringKeySerializer());
+           activeAllocationIds = DiffableUtils.diff(before.activeAllocationIds,
    ↪ after.activeAllocationIds,
+                   DiffableUtils.getVIntKeySerializer(), DiffableUtils.
    ↪ StringSetValueSerializer.getInstance());
        }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as in change B.2.19.

Code level perspective. In the listing below, a variable `primaryTerms` was added to identify operations that are coming from a failed primary shard when creating new *diff* for a cluster state.

```
        public IndexMetaDataDiff(IndexMetaData before, IndexMetaData after) {
            ...
            version = after.version;
            state = after.state;
            settings = after.settings;
+           primaryTerms = after.primaryTerms;
            mappings = DiffableUtils.diff(before.mappings, after.mappings);
            aliases = DiffableUtils.diff(before.aliases, after.aliases);
            customs = DiffableUtils.diff(before.customs, after.customs);
        }
```

B.2.32 Conflict 32

Category: Change of Method call or object creation

Mainline Change

Category: Feature enhancement

Project level perspective. The cluster health status change, from red to green or yellow, from green to yellow or red, from yellow to red or green, and vice versa, was logged for tracing or debugging purpose. The cluster health changes are logged on info level. The log messages can be logged in six levels depending on the severity of a situation or action. The log levels are debug, error, info, fatal, trace and warn. The info level is used to log messages that are generally important for

normal operation of an application or system.

Code level perspective. Listing below shows a change which is part of logging the cluster health status change. A parameter which provide a reason for the change was added to a method `reroute` which reroutes a routing table, that provide mapping between nodes and its shards and indexes they host, using live nodes as a criterion. In the test, the parameter value `"reroute"` is passed as the reason for the cluster health change when rerouting the routing table.

```
      public void
          ↪ testBackupElectionToPrimaryWhenPrimaryCanBeAllocatedToAnotherNode() {
          ...
          logger.info("Adding third node and reroute and kill first node");
          clusterState = ClusterState.builder(clusterState).nodes(DiscoveryNodes.
              ↪ builder(clusterState.nodes()).put(newNode("node3")).remove("node1"
              ↪ )).build();
          prevRoutingTable = routingTable;
-         routingTable = strategy.reroute(clusterState).routingTable();
+         routingTable = strategy.reroute(clusterState, "reroute").routingTable();
          clusterState = ClusterState.builder(clusterState).routingTable(
              ↪ routingTable).build();
          routingNodes = clusterState.getRoutingNodes();
          ...
      }
```

Branch Change

Category: Feature introduction

Project level perspective. Same as change B.2.19 which introduced the primary terms to track number of replica shard promotion to primary shard.

Code level perspective. The listing below shows changes which were made as part of introducing primary terms. The listing shows test on retaining a primary shard when the primary can be allocated to another node during rerouting. In the listing, a current routing table `routingTable` is compared with a previous routing table `prevRoutingTable` routing table.

```
      public void
          ↪ testBackupElectionToPrimaryWhenPrimaryCanBeAllocatedToAnotherNode() {
          ...
          logger.info("Adding third node and reroute and kill first node");
          clusterState = ClusterState.builder(clusterState).nodes(DiscoveryNodes.
              ↪ builder(clusterState.nodes()).put(newNode("node3")).remove("node1"
              ↪ )).build();
-         prevRoutingTable = routingTable;
-         routingTable = strategy.reroute(clusterState).routingTable();
-         clusterState = ClusterState.builder(clusterState).routingTable(
    ↪ routingTable).build();
+         RoutingTable prevRoutingTable = clusterState.routingTable();
+         result = strategy.reroute(clusterState);
+         clusterState = ClusterState.builder(clusterState).routingResult(result).
    ↪ build();
          routingNodes = clusterState.getRoutingNodes();
+         routingTable = clusterState.routingTable();

          assertThat(prevRoutingTable != routingTable, equalTo(true));
          ...
      }
```

L

B.2.33 Conflict 33

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Dependency injector on the shard level was removed and replaced with constructor calls.

Code level perspective. The listing below shows changes which are part of replacing injector with constructor calls on the shard level. The `HashMap` of type `IndexShardInjectorPair` was changed to `IndexShard` and a variable `indexShardInjectorPair` which holds shard indexing instance and its injector was removed. Furthermore, the shard injector `shardInjector` passed to the method `closeShard`, which closes shard so that not to allow operations on it when the changes on the engine are rolled back, was changed to `indexShard` which is an instance variable of shard indexing.

```
     public synchronized void removeShard(int shardId, String reason) {
         ...
         logger.debug("[{}] closing... (reason: [{}])", shardId, reason);
-        HashMap<Integer, IndexShardInjectorPair> tmpShardsMap = new HashMap<>(
    ↪ shards);
-        IndexShardInjectorPair indexShardInjectorPair = tmpShardsMap.remove(
    ↪ shardId);
-        indexShard = indexShardInjectorPair.getIndexShard();
-        shardInjector = indexShardInjectorPair.getInjector();
+        HashMap<Integer, IndexShard> tmpShardsMap = new HashMap<>(shards);
+        indexShard = tmpShardsMap.remove(shardId);
         shards = ImmutableMap.copyOf(tmpShardsMap);
-        closeShardInjector(reason, sId, shardInjector, indexShard);
+        closeShard(reason, sId, indexShard, indexShard.store());
         logger.debug("[{}] closed (reason: [{}])", shardId, reason);
     }
```

Branch Change

Category: Library removal

Project level perspective. An immutable map, which maps values from keys and cannot be changed, was removed in the Elasticsearch project codebase. The change was part of removing Guava[2], which is a collection of Java core libraries. The immutable map was part of the collection.

Code level perspective. The changes in the listing below are part of removing immutable map. The immutable map `shards` was modified to use method `unmodifiableMap` from Java collections class as `ImmutableMap` class was removed from the project. Furthermore, the variable `tmpShardsMap` was renamed to `newShards`.

```
     public synchronized void removeShard(int shardId, String reason) {
```

[2]https://github.com/google/guava

```
            ...
            logger.debug("[{}] closing... (reason: [{}])", shardId, reason);
-           HashMap<Integer, IndexShardInjectorPair> tmpShardsMap = new HashMap<>(
    ↪ shards);
-           IndexShardInjectorPair indexShardInjectorPair = tmpShardsMap.remove(
    ↪ shardId);
+           HashMap<Integer, IndexShardInjectorPair> newShards = new HashMap<>(shards
    ↪ );
+           IndexShardInjectorPair indexShardInjectorPair = newShards.remove(shardId)
    ↪ ;
            indexShard = indexShardInjectorPair.getIndexShard();
            shardInjector = indexShardInjectorPair.getInjector();
-           shards = ImmutableMap.copyOf(tmpShardsMap);
+           shards = unmodifiableMap(newShards);
            closeShardInjector(reason, sId, shardInjector, indexShard);
            logger.debug("[{}] closed (reason: [{}])", shardId, reason);
        }
```

B.2.34 Conflict 34

Category: Change of an assert statement Expression

Mainline Change

Category: Refactoring

Project level perspective. Query parsing exception was renamed to parsing exception which is more generic so that it can be reused rather than creating subclasses.

Code level perspective. Listing below shows a change which was part of renaming query parsing exception to parsing exception. The class `QueryParsingException` was renamed to `ParsingException`.

```
    public void testPercolatorUpgrading() throws Exception {
        ...
        try {
            client().prepareIndex("test2", PercolatorService.TYPE_NAME)
                    .setSource(jsonBuilder().startObject().field("query",
                        ↪ termQuery("field1", "value")).endObject()).get();
            fail();
        } catch (PercolatorException e) {
            e.printStackTrace();
-           assertThat(e.getRootCause(), instanceOf(QueryParsingException.class))
    ↪ ;
+           assertThat(e.getRootCause(), instanceOf(ParsingException.class));
        }
    }
```

Branch Change

Category: Refactoring

Project level perspective. Context for query parsing was separated to have two different steps for query parsing and shard querying. The change was made as it might be required to run the steps in separate phases such as on shards and master node. The query parsing context is used to parse xContent from the request and

shard querying context is used when creating Lucene query on the shard level.

Code level perspective. The listing below shows part of a change to separate the query parsing context. The class `QueryParsingException` which is used when parsing queries from the given query parsing context was renamed to `QueryShardException`.

```
    public void testPercolatorUpgrading() throws Exception {
        ...
        try {
            client().prepareIndex("test2", PercolatorService.TYPE_NAME)
                    .setSource(jsonBuilder().startObject().field("query",
                        ↪ termQuery("field1", "value")).endObject()).get();
            fail();
        } catch (PercolatorException e) {
            e.printStackTrace();
-           assertThat(e.getRootCause(), instanceOf(QueryParsingException.class))
    ↪ ;
+           assertThat(e.getRootCause(), instanceOf(QueryShardException.class));
        }
    }
```

B.2.35 Conflict 35

Category: Addition of statements in the Same area

Mainline Change

Category: Refactoring

Project level perspective. The `score type` field which was used in `has child` and `has parent` joining queries when querying child and parent documents respectively in a query Project level specific language (DSL) to specify a query score type (`max`, `min`, `avg`, `sum` or `none`), was replaced with the `score mode` field. The `score mode` field exist in Lucene, therefore it was favored instead of the `score type` field. Furthermore, in this change, the score type (score mode) `sum` was replaced with `total` which also exist in the Lucene.

Code level perspective. In the listing below, a method `scoreType` was renamed to `scoreMode`.

```
    protected void doXContent(XContentBuilder builder, Params params) throws
        ↪ IOException {
        builder.startObject(HasParentQueryParser.NAME);
        builder.field("query");
        queryBuilder.toXContent(builder, params);
        builder.field("parent_type", parentType);
-       if (scoreType != null) {
-           builder.field("score_type", scoreType);
+       if (scoreMode != null) {
+           builder.field("score_mode", scoreMode);
        }
        if (boost != 1.0f) {
            builder.field("boost", boost);
        }
        ...
    }
```

Branch Change

Category: Refactoring

Project level perspective. Parsing of an index search query for `has child` query, which query child documents, was refactored to separate its parsing in the Elasticsearch and Lucene query creation. First, the query is parsed to create an xContent object which can be streamed. Then, the object is converted to the Lucene query. This change allows among other things to easily test the creation of the xContent parsing since the parsing is in one place.

Code level perspective. The listing below shows a change which refactored `has child` query parsing. As part of separating parsing of the index search query for the `has child` query, the method `doXContent` was refactored to remove parsing for the Lucene query creation. A new method `doToQuery` was added forr the removed parsing for creating the Lucene query.

```
-    protected void doXContent(XContentBuilder builder, Params params) throws
    ↪ IOException {
-        builder.startObject(HasParentQueryParser.NAME);
-        builder.field("query");
-        queryBuilder.toXContent(builder, params);
-        builder.field("parent_type", parentType);
-        if (scoreType != null) {
-            builder.field("score_type", scoreType);
+    protected Query doToQuery(QueryShardContext context) throws IOException {
+        Query innerQuery = query.toQuery(context);
+        if (innerQuery == null) {
+            return null;
         }
-        if (boost != 1.0f) {
-            builder.field("boost", boost);
+        innerQuery.setBoost(boost);
+        DocumentMapper parentDocMapper = context.mapperService().documentMapper(
    ↪ type);
+        if (parentDocMapper == null) {
+            throw new QueryParsingException(context.parseContext(), "[has_parent]
    ↪  query configured 'parent_type' [" + type
+                    + "] is not a valid type");
         }
-        if (queryName != null) {
-            builder.field("_name", queryName);
         ...
+    }
+
+    @Override
+    protected void doXContent(XContentBuilder builder, Params params) throws
    ↪ IOException {
+        builder.startObject(NAME);
+        builder.field("query");
+        query.toXContent(builder, params);
+        builder.field("parent_type", type);
+        builder.field("score", score);
+        printBoostAndQueryName(builder);
         if (innerHit != null) {
-            builder.startObject("inner_hits");
-            builder.value(innerHit);
-            builder.endObject();
+            innerHit.toXContent(builder, params);
         }
         builder.endObject();
     }
```

B.2.36 Conflict 36

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. Same as B.2.35 which replaced `score type` field with `score mode` in joining queries when querying a document.

Code level perspective. The listing below shows a change which replaced a score mode `sum` with `total` in a `has child` joining query search test.

```
      public void testMinMaxChildren() throws Exception {
          ...
          assertThat(response.getHits().hits()[2].id(), equalTo("2"));
          assertThat(response.getHits().hits()[2].score(), equalTo(1f));

-         response = minMaxQuery("sum", 0, 3);
+         response = minMaxQuery("total", 0, 3);

          assertThat(response.getHits().totalHits(), equalTo(3l));
          assertThat(response.getHits().hits()[0].id(), equalTo("4"));
          ...
      }
```

Branch Change

Category: Refactoring

Project level perspective. Same as B.2.35 which refactored parsing of an index search query.

Code level perspective. Listing below shows a change which was part of the `has child` query parsing refactoring. A score type `SUM` was fetched from a class `ScoreType` which defines mapping of the child documents into a parent document.

```
      public void testMinMaxChildren() throws Exception {
          ...
          assertThat(response.getHits().hits()[2].id(), equalTo("2"));
          assertThat(response.getHits().hits()[2].score(), equalTo(1f));

-         response = minMaxQuery("sum", 0, 3);
+         response = minMaxQuery(ScoreType.SUM, 0, 3);

          assertThat(response.getHits().totalHits(), equalTo(3l));
          assertThat(response.getHits().hits()[0].id(), equalTo("4"));
          ...
      }
```

B.2.37 Conflict 37

Category: Change of IF statement condition

Mainline Change

Category: Library removal

Project level perspective. Objects library for Java which is a helper functions used on any object was removed from the Elasticsearch project codebase. The library is part of Google's Guava core libraries for Java. Similar to B.2.33, this change was part removing the Guava library from the Elasticsearch codebase.

Code level perspective. Listing below shows a change which was part of removing Google's Guava Objects library. A method `equal` from the Guava Objects library was changed to `equals` which is a common method for Java object.

```
    private void publish(LocalDiscovery[] members, ClusterChangedEvent
        ↪ clusterChangedEvent, final BlockingClusterStatePublishResponseHandler
        ↪ publishResponseHandler) {
        ...
        try {
             for (final LocalDiscovery discovery : members) {
                 if (nodeSpecificClusterState.nodes().localNode() != null) {
                     ...
                     discovery.clusterService.submitStateUpdateTask("local-disco-
                         ↪ receive(from master)", new
                         ↪ ProcessedClusterStateNonMasterUpdateTask() {
                         @Override
                         public ClusterState execute(ClusterState currentState) {
-                            if (nodeSpecificClusterState.version() < currentState
    ↪ .version() && Objects.equal(nodeSpecificClusterState.nodes().masterNodeId()
    ↪ , currentState.nodes().masterNodeId())) {
+                            if (nodeSpecificClusterState.version() < currentState
    ↪ .version() && Objects.equals(nodeSpecificClusterState.nodes().masterNodeId
    ↪ (), currentState.nodes().masterNodeId())) {
                                 return currentState;
                             }
                     ...
                         });
                 } else {
                     publishResponseHandler.onResponse(discovery.localNode);
                 }
             }
             ...
        } catch (Exception e) {
             // failure to marshal or un-marshal
             throw new IllegalStateException("Cluster state failed to serialize", e
                 ↪ );
        }
    }
```

Branch Change

Category: Refactoring

Project level perspective. Cluster state queue from Zen discovery module which is used for unicast discovery of nodes and selection of a master node in the cluster was refactored so that to provide a buffer for processed cluster states which are not committed yet. The buffer are used to holds the cluster state, then are sent to the Zen discovery module after they are committed. The buffer allows the cluster state to be committed even there is a new cluster state that has been processed.

Code level perspective. The listing below shows a change which was part of refactoring cluster state queue in the Zen discovery module. The conditions `nodeSpecificClusterState.version() < currentState.version()` and `Objects.equal( nodeSpecificClusterState.nodes().masterNodeId(), currentState.nodes() .masterNodeId())`, which check if a current cluster state's version is greater than a cluster state's version for specific node and compare if an object of the cluster state for the specific node is equal to an object of the current cluster state respectively, were replaced with `currentState.supersedes(nodeSpecificClusterState)`, which check if the current cluster state replaces the cluster state's for the specific node.

```
    private void publish(LocalDiscovery[] members, ClusterChangedEvent
        ↪ clusterChangedEvent, final BlockingClusterStatePublishResponseHandler
        ↪ publishResponseHandler) {
        ...
        try {
             for (final LocalDiscovery discovery : members) {
                 if (nodeSpecificClusterState.nodes().localNode() != null) {
                     ...
                     discovery.clusterService.submitStateUpdateTask("local-disco-
                         ↪ receive(from master)", new
                         ↪ ProcessedClusterStateNonMasterUpdateTask() {
                         @Override
                         public ClusterState execute(ClusterState currentState) {
-                            if (nodeSpecificClusterState.version() < currentState
    ↪ .version() && Objects.equal(nodeSpecificClusterState.nodes().masterNodeId()
    ↪ , currentState.nodes().masterNodeId())) {
+                            if (currentState.supersedes(nodeSpecificClusterState)
    ↪ ) {
                                 return currentState;
                             }
                     ...
                     });
                 } else {
                     publishResponseHandler.onResponse(discovery.localNode);
                 }
             }
             ...
        } catch (Exception e) {
             // failure to marshal or un-marshal
             throw new IllegalStateException("Cluster state failed to serialize", e
                 ↪ );
        }
    }
```

B.2.38 Conflict 38

Category: Change of Method call or object creation

Mainline Change

Category: Refactoring

Project level perspective. The class loader in the common settings which is used to load resources or classes was removed. The common settings implement creation of the xContent. The change was made to prevent errors as the common settings can be accessed by any plugin or internal API. With this change, the resources or classes are now loaded using normal Java methods such as `getClass().getResource()`.

Code level perspective. The listing below shows changes made as part of removing the class loader in the common settings. The method `loadFromStream` is used to load a file instead of `loadFromClasspath`.

```
    public void testSynonymsAnalysis() throws IOException {
+       String json = "/org/elasticsearch/index/analysis/synonyms/synonyms.json";
        Settings settings = settingsBuilder().
-               loadFromClasspath("org/elasticsearch/index/analysis/synonyms/
    ↪ synonyms.json")
+               loadFromStream(json, getClass().getResourceAsStream(json))
                .put("path.home", createTempDir().toString())
                .put(IndexMetaData.SETTING_VERSION_CREATED, Version.CURRENT).
                    ↪ build();
        ...
    }
```

Branch Change

Category: Test improvement

Project level perspective. The method which resolve configuration files path was removed. The method resolves the configuration files by looking into class path, configuration directory and then prefixes configuration directory in the class path. The change corrects the setup of the fake home directories for test configuration files.

Code level perspective. The listing below shows changes which fix setting up fake home directories for test configuration files. The home directory path `home` is created using temporary directory by method `createTempDir`. Then the `home` directory is resolved to configuration path `config`. Thereafter, the configuration files are added to the `config` path.

```
    public void testSynonymsAnalysis() throws IOException {
+       InputStream synonyms = getClass().getResourceAsStream("synonyms.txt");
+       InputStream synonymsWordnet = getClass().getResourceAsStream("
    ↪ synonyms_wordnet.txt");
+       Path home = createTempDir();
+       Path config = home.resolve("config");
+       Files.createDirectory(config);
+       Files.copy(synonyms, config.resolve("synonyms.txt"));
+       Files.copy(synonymsWordnet, config.resolve("synonyms_wordnet.txt"));
+
        Settings settings = settingsBuilder().
                loadFromClasspath("org/elasticsearch/index/analysis/synonyms/
                    ↪ synonyms.json")
-               .put("path.home", createTempDir().toString())
+               .put("path.home", home)
                .put(IndexMetaData.SETTING_VERSION_CREATED, Version.CURRENT).
                    ↪ build();

        Index index = new Index("test");
        ...
    }
```

B.2.39 Conflict 39

Category: Modification and removal of statements

Mainline Change

Category: Refactoring

Project level perspective. The dependency on Java serialization which was used to serialize exceptions was removed to prevent problems encountered when upgrading Java Virtual Machines (JVM). The change removed the usage of `ObjectInputStream` and `ObjectOutputStream` to serialize objects and added custom serialization of exceptions.

Code level perspective. Listing below shows changes which add custom serialization of Elasticsearch exceptions. If there is a byte in the stream, a `key` is assigned to the integer stored in the variable-length format. Then for each key, appropriate exception is returned.

```
     public <T extends Throwable> T readThrowable() throws IOException {
-        try {
-            ObjectInputStream oin = new ObjectInputStream(this);
-            return (T) oin.readObject();
-        } catch (ClassNotFoundException e) {
-            throw new IOException("failed to deserialize exception", e);
+        if (readBoolean()) {
+            int key = readVInt();
+            switch (key) {
+                case 0:
+                    final String name = readString();
+                    return (T) readException(this, name);
+                case 1:
+                    // this sucks it would be nice to have a better way to
    ↪ construct those?
+                    String msg = readOptionalString();
+                    final int idx = msg.indexOf(" (resource=");
+                    final String resource = msg.substring(idx + " (resource=".
    ↪ length(), msg.length()-1);
+                    msg = msg.substring(0, idx);
+                    return (T) readStackTrace(new CorruptIndexException(msg,
    ↪ resource, readThrowable()), this); // Lucene 5.3 will have getters for all
    ↪ these
+                case 2:
+                    return (T) readStackTrace(new IndexFormatTooNewException(
    ↪ readOptionalString(), -1, -1, -1), this);  // Lucene 5.3 will have getters
    ↪ for all these
+                case 3:
+                    return (T) readStackTrace(new IndexFormatTooOldException(
    ↪ readOptionalString(), -1, -1, -1), this);  // Lucene 5.3 will have getters
    ↪ for all these
+                case 4:
+                    return (T) readStackTrace(new NullPointerException(
    ↪ readOptionalString()), this);
+                case 5:
+                    return (T) readStackTrace(new NumberFormatException(
    ↪ readOptionalString()), this);
+                case 6:
+                    return (T) readStackTrace(new IllegalArgumentException(
    ↪ readOptionalString(), readThrowable()), this);
+                case 7:
+                    return (T) readStackTrace(new IllegalStateException(
    ↪ readOptionalString(), readThrowable()), this);
+                case 8:
+                    return (T) readStackTrace(new EOFException(readOptionalString
    ↪ ()), this);
+                case 9:
+                    return (T) readStackTrace(new SecurityException(
    ↪ readOptionalString(), readThrowable()), this);
+                case 10:
```

```
+                    return (T) readStackTrace(new StringIndexOutOfBoundsException
    ↪ (readOptionalString()), this);
+                case 11:
+                    return (T) readStackTrace(new ArrayIndexOutOfBoundsException(
    ↪ readOptionalString()), this);
+                case 12:
+                    return (T) readStackTrace(new AssertionError(
    ↪ readOptionalString(), readThrowable()), this);
+                case 13:
+                    return (T) readStackTrace(new FileNotFoundException(
    ↪ readOptionalString()), this);
+                case 14:
+                    final String file = readOptionalString();
+                    final String other = readOptionalString();
+                    final String reason = readOptionalString();
+                    readOptionalString(); // skip the msg - it's composed from
    ↪ file, other and reason
+                    return (T) readStackTrace(new NoSuchFileException(file, other
    ↪ , reason), this);
+                case 15:
+                    return (T) readStackTrace(new OutOfMemoryError(
    ↪ readOptionalString()), this);
+                case 16:
+                    return (T) readStackTrace(new AlreadyClosedException(
    ↪ readOptionalString(), readThrowable()), this);
+                case 17:
+                    return (T) readStackTrace(new LockObtainFailedException(
    ↪ readOptionalString(), readThrowable()), this);
+                default:
+                    assert false : "no such exception for id: " + key;
+            }
        }
+       return null;
    }
```

Branch Change

Category: Refactoring

Project level perspective. A new abstraction was introduced to serialize queries by writing the current object into the output stream rather than their Json, and deserialize queries by using their names. This change, similar to B.2.35, was part of the index query refactoring to separate its parsing in the Elasticsearch and query creation in the Lucene.

Code level perspective. The listing below shows changes which were part of abstracting serialization of the index queries. The returned deserialization `(T) oin.readObject()` was refactored to an object, then the object was returned.

```
    public <T extends Throwable> T readThrowable() throws IOException {
        try {
            ObjectInputStream oin = new ObjectInputStream(this);
-           return (T) oin.readObject();
+           @SuppressWarnings("unchecked")
+           T object = (T) oin.readObject();
+           return object;
        } catch (ClassNotFoundException e) {
            throw new IOException("failed to deserialize exception", e);
        }
    }
```

B.2.40 Conflict 40

Category: Changes in Different statements in the same area

Mainline Change

Category: Bug fix

Project level perspective. Support for missing fields in index queries when serializing the queries in Json format was fixed. The wrong logic implemented in some Java API builders resulted to missing fields such as `_name` which is used to assign or tag a name to the query.

Code level perspective. The listing below shows a change which fixes the wrong logic to support the assingment of names to the index queries. The condition `queryName != null` was changed to `queryName == null`. The change allows the the xContent builder to buid `_name` field.

```
    public void doXContent(XContentBuilder builder, Params params) throws
        ↪ IOException {
        builder.startObject(PrefixQueryParser.NAME);
-       if (boost == -1 && rewrite == null && queryName != null) {
+       if (boost == -1 && rewrite == null && queryName == null) {
            builder.field(name, prefix);
        } else {
            builder.startObject(name);
            builder.field("prefix", prefix);
            if (boost != -1) {
                builder.field("boost", boost);
            }
            if (rewrite != null) {
                builder.field("rewrite", rewrite);
            }
            if (queryName != null) {
                builder.field("_name", queryName);
            }
            builder.endObject();
        }
        builder.endObject();
    }
```

Branch Change

Category: Refactoring

Project level perspective. The index query parser name which identify the parsers to its corresponding query builder was moved from the parsers to the builders. The change was made to link parser and builder implementations without necessary converting queries to the xContent builders and to ensure the index query parser names are unique.

Code level perspective. The listing below shows a change which was part of moving the index parser name to the query builder. The query parser name `PrefixQueryParser.NAME` was changed to `NAME` which was set in the builder.

```
    public void doXContent(XContentBuilder builder, Params params) throws
        ↪ IOException {
-       builder.startObject(PrefixQueryParser.NAME);
+       builder.startObject(NAME);
        if (boost == -1 && rewrite == null && queryName != null) {
            builder.field(name, prefix);
        } else {
            builder.startObject(name);
            builder.field("prefix", prefix);
            if (boost != -1) {
                builder.field("boost", boost);
            }
            if (rewrite != null) {
                builder.field("rewrite", rewrite);
            }
            if (queryName != null) {
                builder.field("_name", queryName);
            }
            builder.endObject();
        }
        builder.endObject();
    }
```