Studying Pull Request Merges: A Case Study of Apache Spark

Alaleh Hamidi[1], Arghavan Sanei[1], Iman Bavandsavadkouhi[1], Ghoncheh FahimiMoghaddam[1]
[1] *Dep. of Computer Science and Engineering, Polytechnique Montreal, Canada*

***Abstract*—GitHub is one of the most important repositories for sharing code between developers. In this project, we report two studies that investigated in Pull Request (PR) merges of Apache Spark, an open-source distributed general cluster-computing framework. One of the major factors influencing the PR merge time is merge type. According to our study, while there are different types of merging, Apache Spark developers chose squash more than the other types, almost 44%. Besides, metrics like affiliation with Apache PR and number of commits, comments and author comments affect the review time and review outcome.**

I. Introduction

GitHub is one of the ideal places for sharing code between developers and doing research on different code repositories which can be called as a social networking website for developers as the aim of version controlling, managing, and storing revision of the projects [1]. Although, there is no sufficient study for available data from GitHub to describe the quality and properties of it, it is one of the important source software artifacts [2].

When developers make some changes in code, they can present them by "pull request". Pull-based development is one of the ways of developing distributed projects. In such a way that the developer's contributions are pulled from forked repositories, modified, and then later merged back into the main repository [3]. Therefore, it is a good place to analyze the changes which have been done on that specific repository [1]. Then the owner of repository make the decision whether let the files changed merge or not into the codebase [3].

According to different studies there are some pull requests that the data gathered from their repositories on GitHub displays that they have not been merged into the main branch [2].There are three different types of merging :

1) GitHub merges: this type of merging uses the GitHub facility.
2) Cherry-pick merge: this type of merge happens when a developer decides to commit a subset of changes to the repository.
3) Commit squashing: when the developer by one commit can commit all of them to make additional changes.

Apache Spark is an open source general engine for big data processing on GitHub [4]. We use this repository to extract data and find several factors in each pull request that was closed and finding the merged PRs we also perform the statistical models that indicate both PR review time and review outcome are influenced by number of commits, number of comments, number of author comments and affiliation with Apache. There are some factors which play a critical role in merging the new file changes into the Spark repository. As a result, we apply some data ming approaches on Apache Spark to identify the type of its merges. Further, we did some manual inspection on finding the type of merges. Then, we investigate which factors make influences on the time and end results of merging the pull request. First, we apply data mining techniques on the Apache spark repository on GitHub to find merge PR. We also investigate what factors contribute to PR merge time and outcome. In the second study, we perform a qualitative analysis of the results of our first study. The results provide insights into how these developers perform pull request merges, and what factors they find contribute to how they review and merge pull requests [3].

II. Approach

To estimate the results of original paper, we replicate the its approach for the Apache Spark repository which is an open-source distributed framework. The project is hosted on GitHub and employs a pull based method for submitting and accepting code contributions and every authorized developer can comment on each PR.

A. Data Collection

We should extract the data related to the variables listed in Table I same as the original paper. To obtain our required data from Spark repository, we used Rest GitHub API and relevant JSON files. We track each variable in the PR address at Rest GitHub API and then we find the JSON file that was related to the PR, then we wrote a python script for each variable to extract the data. We considered the contributions made to Apache spark from the first day of contribution until November 2019. Since our target PRs are the ones that developers had decided on, we just investigate the closed PRs.

As Apache Spark contains a high number of closed pull requests (about 26,803 PR), we decide to choose a random sample of PRs to work with. This sample must contain at least 379 PRs to be approved as a generalizable sample (Confidence level = 95% and Confidence Interval= 5). So, we get our random sample which contained 399 closed PRs.

In addition to the factors of Table I, we extracted some other factors to get the dependent variables of our study: Review time and IsMereged. To calculate the review time, we get the PR submission date/time and the PR closing date/time. We got PR unique ID as the key factor as well.

Table I: Description of studied factors

Independent Variable	Description
PR size	Sum of added and removed LOC
# of files	# of files changed by a PR
# of commits	# of commits in a PR
PR author experience	# of prior PRs submitted by PR author
# of Comments	# of comments left on a PR
# of author comments	# of comments left by the PR author
# of commenting developers	# of developers participating in discussion
# of in-code comments	# of comments left on source code
# of author in-code comments	# of comments left on source code by author
# of in-code commenting devs	# of developers who commented on source code
PR author's affiliation	An org that a PR author affiliates with

B. Inspection

As Kalliamvakou et al. mentioned, GitHub data is not always reliable regarding whether a PR has been merged or not [2], because PRs can be merged in several different approaches. Kalliamvakou et al. proposed following heuristics to recover "missing" merge flags.

H1: At least one of the commits in the pull request appears in the target project's master branch.

H2: A commit closes the pull request using its log (e.g. if the log of the commit includes one of the closing keywords, see above) and that commit appears in the project's master branch. This means that the pull request commits were squashed onto one commit and this commit was merged.

H3: One of the last 3 (in order of appearance) discussion comments contain a commit unique identifier, this commit appears in the project's master branch and the corresponding comment can be matched by the following regular expression:
(?:merg|appl|pull|push|integrat)(?:ing|i?ed)

H4: The latest comment prior to closing the pull request matches the regular expression above [2].

We tried to apply these heuristics on the PRs of our sample, but unfortunately only the first one was applicable. The reason is that developers of Spark repository don't follow a specific rutin to use some keywords in the comments of each PR and in the messages of each commit.

Also, when we apply the first heuristic on our sample which contain 399 PRs, it returned only one PR as merged type (less than 1%). So, we decided to inspect all the sample manually.

To manual inspection of the selected PRs, we divide sample into two parts and two researchers considered each part separately. To ensure that both persons have the same understanding of merging type, we selected 30 PRs randomly to be labeled by these researchers. Then we compared each labelling result and calculated percent agreement. Percent agreement was 90% because of 3 differences.

C. Data Analysis

We used two different statistical model to analyze our data set. We built Multiple Linear Regression Model (MLR) to investigate which factors affect the review time. In addition, we used Logistic Regression Model (LRM) to determine the factors that influence the review decision (to merge or not).

Data Cleaning: The numbers of observations for all of variables are equal to our sample size, which implies there are no missing values.

Since probability distribution function for each feature should show a normal distribution, and we have a few outliers in the Size(LOC) variable, we removed the 5% of all PRs with the largest Size(LOC) value.

Variable transformation: We checked the linearity using scatter plot between dependent variable and each independent variable. As we could spot linear patterns, there was no need to apply log transformation. Note that this part was done only for MLR model, not the LRM one.

Controlling Multicollinearity: In regression models, we should avoid multicollinearity which is a high correlation among two or more independent variables. Using VIF (Variance Inflation Factor), we can understand how much the variance of each variable is explained by the collinearity with other variables. We used the vif method of statsmodels package in Python. We eliminated the factors which have vif score more than 5.

Standardization means the process of transforming data into a standard scale. The common problem when working with numerical data is difference in magnitudes. Standardization can solve this problem by using the formula below:

Standardized variable = (Original variable- Mean of Original variable) / Standard deviation of Original variable

We used StandardScaler module from Sklearn package of Python.

Model evaluation: To evaluate MLR model, we used Adjusted R2. Accuracy is user for evaluating LRM as well. The amount of these parameters are shown in Table IV.

III. Results

A. RQ1: Merge type and effect on review time

As described in Section 2, we determined PR merge types by applying heuristics and then manually classifying PRs to label all PRs in our dataset. The results of this inspection are presented in Table II.

As we can see in Table II, even lower than 1% of PRs were merged via "Merge Button" of native GitHub UI. It means that there are only very small number of PRs which need no further change. The most popular merge type was squashing.

To analyze the effect of merge type on review time, we used two kinds of statistical tests: Kruskal-Wallis analysis of variance [5] and Mann-Whitney U (MWW) [6].

Kruskal-Wallis: We employ this test to determine if merge type has a statistically significant effect on review time or not. Since we assume α= 0.05, the amount of Critical value becomes 7.8144. This test exposed that merge type has a

Table II: Classification of PRs by Merge type

	Count	Percent	Sum (Review Time)	Median (Review Hours)
GitHub	3	0.75	41.12833334	13.70944445
Cherry-picking	107	26.88	1149.410833	10.74215732
Squashing	175	43.96	1973.691944	11.27823968
Not Merged	113	28.93	1005.688333	8.8998

statistically significant effect on review time, because the amount of X2(3) (which is about 9.77) is greater than Critical Value.

Mann-Whitney U test (MWW): We apply this test to show where the significant difference occurs. According to the calculated parameters listed in Table III and comparing the amount of U-Statistics with the critical value (Significant Level = 0.1), there are significant difference among all pairs except between GitHub and Cherry-picking.

Table III: U-test parameters for each merge type

	Sum of Rank	Count	U Statistic
GitHub	726	3	720
Cherry-picking	22259.5	107	16481.5
Squashing	37058	175	21658
Not Merged	19357.5	113	12916.5

RQ1 Results: While most developers merge pull request via squashing (43.96%), Cherry-pick PR merge type is also a common practice. Merge type has a statistically significant effect on PR merge time; Merges done via the GitHub UI takes more time and is used less than the others.

B. RQ2: Factors affecting merge time/decision

We built Multiple Linear Regression Model (MLR) to investigate which factors affect the review time. In addition, we used Logistic Regression Model (LRM) to determine the factors that influence the review decision (to merge or not). Both models used the factors listed in Table1 as independent variables. But merge types factor, obtained from manual inspection, is used for MLR as well. Because of correlation between merge type and the dependent variable of LRM, we didn't consider this factor in LRM. We calculated the weight of each factor for MLR to cover the property of both coefficient and p-value. Table IV represent a brief description of these two models' result.

Review time: The MLR model indicates that number of commits has a statistically significant effect on review time. Because of positive value of its coefficient, the greater number of commits are, the longer the PR takes for developers to review it.

Similarly, number of comments, number of commenting developers and number of author comments have a positive coefficient. So, each comment can increase review time as it needs to be read by other developers and judging this comment takes time. Specially, author comments are getting more attention. In addition, we can conclude each developer who is participating in a conversation, cause a delay on PR decision. But the negative coefficient for the number of in-code commenting developers shows that the in-code comments of more developers for each PR, make PR clearer to decide.

The model shows that PRs submitted by developers from Apache receive faster reviews on their PRs.

Surprisingly, GitHub merges don't have any effect on review time, whereas Cherry-picking and Squash merges can decrease the review time.

Merge Decision: The second model (LRM) shows that the greater number of files and number of commits reduce the chance of being merged.

Like the model on review time, coefficient of number of comments is positive that means, PRs which get more comments, have greater probability to be merged, whereas higher number of author comments surprisingly decrease this probability.

The affiliation of PR author with Apache has a notable effect on the decision for merging. Since the coefficient of this variable is positive, PRs that come from the developers of Apache have a higher chance to be accepted.

RQ2 results: The statistical models indicate that both PR review time and review outcome are influenced by number of commits, number of comments, number of author comments and affiliation with Apache.

IV. Comparison of Results

RQ1: Merge type and effect on review time

Both studies show that most PRs are merged via squashing and cherry-pick PR merge type is also a common practice.

Merge type has a statistically significant effect on PR merge time in both studies. This result can be generalized on the other GitHub repositories. The only difference occurs for GitHub merge that might be because of Spark developers' approach to change the PRs even for little issues in messages.

RQ2: Factors affecting merge time/decision

Both studies indicate that affiliation with the respondent company has a significant effect on both review time and review outcome. It means the pull requests submitted by developers from related companies, take less time to be decided and more accepted.

V. Conclusion

As most of the results of each study on different repositories (Shopify and Spark) are not the same, we can conclude that the factors which affect on review time and review outcome, depend on the type of repository. So, we can't generalize the results of these two studies to other GitHub repositories.

Table IV: Models for fitting data

	PR review time		PR review outcome
	Adjusted R^2:0.22		Accuracy: 0.73
	Weight	**Coefficient**	**Coefficient**
Size (LOC)	**	**	**
Number of files	**	**	-0.0300
Number of commits	3.0147	0.5578	-0.0605
Writer experience	**	**	**
# of comments	2.6507	0.1784	0.1076
# of commenting devs	2.0170	1.5053	**
# of author comments	2.9233	0.5813	-0.1535
# of in-code comments	*	*	*
# of in-code commenting devs	-2.1164	-1.3381	**
# of in-code author comments	*	*	*
Affiliation with Apache	-1.2100	-2.7346	0.9763
Affiliation with Spark	**	**	**
Cherry-pick merge	-1.2950	-2.9455	n/a
GitHub merge	**	**	n/a
Squashing merge	-0.6649	-1.3373	n/a
** Removed during VIF analysis*			
***Removed according P-Values*			

REFERENCES

[1] [Online]. Available: https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/

[2] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," *In Proc. of the Working Conf. on mining Soft. Repositories. ACM*, pp. 92–101, 2014.

[3] "Studying pull request merges:a case study of shopify's active merchant."

[4] [Online]. Available: https://www.datacamp.com/community/tutorials/apache-spark-tutorial-machine-learning

[5] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journ. of the American Statistical Ass*, vol. 47, pp. 583–621, 1952.

[6] E. Lehmann and H. D'Abrera, "Nonparametrics: statistical methods based on ranks," 2006.