Utilization of Version Control System for the Development of HPC Software

Final Project Report
by

Chunyan Tang
Kapil Agrawal
Sadika Amreen

CS 494/594 – Fundamentals of Digital Archaeology

Department of Electrical Engineering and Computer Science
The University of Tennessee Knoxville

Fall 2014

Abstract

HPC software engineering is the practice of enhancing software to enable it to perform better on modern multicore processors, GPUs, HPC clusters and supercomputers. In our project we present our analysis of utilization of Version Control System on these developments. The advantage of version control software is that it keeps a track of the complete file history (why/who/what changed), allows for shared development, and supports complex workflows (multiple branches). The analysis on how well one has utilized version controlling on HPC is not been fully studied. Our goal is to find out measures and trends, of these developments made over a period of time.

Table of Contents

Acknowledgement **4**

Introduction **5**

Data Retrieval and Analysis **5**

Methodology *6*

Quality Measure *6*

Predictors *6*

Results and Visualization **7**

Commit Quality *7*

Delta and Size per Commit *8*

Commits per Author *9*

Conclusion **10**

Acknowledgement

We take this opportunity to express our gratitude and deep regards to our instructor Dr. Audris Mockus for his guidance, monitoring and constant encouragement throughout the course.

Introduction

Our project was to analyze utilization of version control system for the development of HPC software. Understanding how HPC code is developed in terms of various predictors such as commit size, frequency of commits, commits per user etc. and suggesting better practices of commits along with tools that utilizes such data to help HPC software development.

As per the proposal we decided to have a comparison between HPC and non-HPC projects in terms of utilization of version control on software development. We decided to retrieve our data from project management sites like Github and Bitbucket; however, we focused only on HPC projects as classifying what is HPC and what is non-HPC is still a big question and project management site like Github and Bitbucket have huge number of *repos* (projects) which runs in hundreds of thousands in numbers.

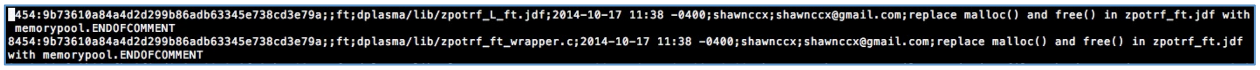
For our analysis we were lucky to get log files of few HPC projects from Innovative Computing Laboratory (ICL) at UTK, and non-HPC log files from Bitbucket. We wish to build a *quality of commit* relation for these projects over the length of the time in which they were developed. Along with some basic information about whether or not commit size is consistent with stable commits, number of contributors, number of commits over the years etc.

Data Retrieval and Analysis

We started with pulling out HPC repos from Bitbucket with most popular keywords that were used by HPC community. Keywords were divided into three categories namely [1] Libraries (which include most of the mathematical, numeric etc), [2] Software Packages (NAMD, Charm, LAMMPS) and [3] generic keywords (quantum chemistry, Parallel Molecular Dynamics). We looked at the description tag and repositories names to match against these keywords. However, the number of repositories that we got was little discouraging to go ahead with our analysis (i.e. around 80). The total number of repositories on Bitbucket came out to be around 400,000. Now having a comparison between HPC and non-HPC on this basis was never a good measure. We finally decided to go ahead with the log files that were provided to us by ICL at UTK and few non-HPC from Bitbucket based on their popularity like most watched, most forked.

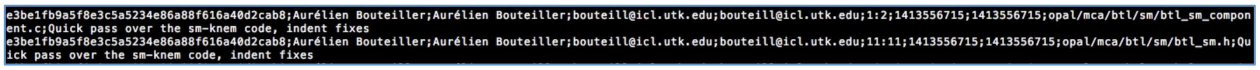
These projects (HPC/non_HPC) were implemented using different version control. For HPC we had to work with Git, Mercurial (hg) and SVN and for non-HPC we had Git and Mercurial. The layout of the log files are shown below:

For HPC



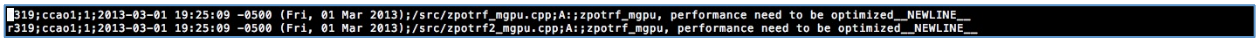
```
454:9b73610a84a4d2d299b86adb63345e738cd3e79a;;ft;dplasma/lib/zpotrf_L_ft.jdf;2014-10-17 11:38 -0400;shawnccx;shawnccx@gmail.com;replace malloc() and free() in zpotrf_ft.jdf with
 memorypool.ENDOFCOMMENT
8454:9b73610a84a4d2d299b86adb63345e738cd3e79a;;ft;dplasma/lib/zpotrf_ft_wrapper.c;2014-10-17 11:38 -0400;shawnccx;shawnccx@gmail.com;replace malloc() and free() in zpotrf_ft.jdf
with memorypool.ENDOFCOMMENT
```

Figure 1.1 Log file layout of Parsec project having Mercurial (hg) as Version Control



```
e3be1fb9a5f8e3c5a5234e86a88f616a40d2cab8;Aurélien Bouteiller;Aurélien Bouteiller;bouteill@icl.utk.edu;bouteill@icl.utk.edu;1:2;1413556715;1413556715;opal/mca/btl/sm/btl_sm_compon
ent.c;Quick pass over the sm-knem code, indent fixes
e3be1fb9a5f8e3c5a5234e86a88f616a40d2cab8;Aurélien Bouteiller;Aurélien Bouteiller;bouteill@icl.utk.edu;bouteill@icl.utk.edu;11:11;1413556715;1413556715;opal/mca/btl/sm/btl_sm.h;Qu
ick pass over the sm-knem code, indent fixes
```

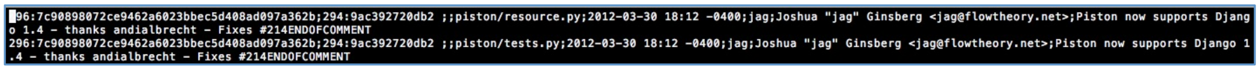
Figure 1.2 Log file layout of Open-mpi project having Git as Version Control



```
319;ccao1;1;2013-03-01 19:25:09 -0500 (Fri, 01 Mar 2013);/src/zpotrf_mgpu.cpp;A:;zpotrf_mgpu, performance need to be optimized__NEWLINE__
r319;ccao1;1;2013-03-01 19:25:09 -0500 (Fri, 01 Mar 2013);/src/zpotrf2_mgpu.cpp;A:;zpotrf_mgpu, performance need to be optimized__NEWLINE__
```

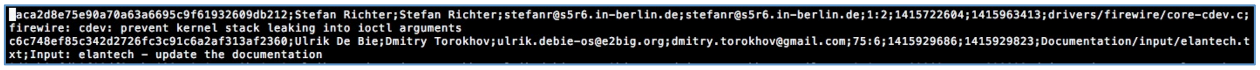
Figure 1.3 Log file layout of MAGMA project having SVN as Version Control

For Non-HPC



```
96:7c90898072ce9462a6023bbec5d408ad097a362b;294:9ac392720db2 ;;piston/resource.py;2012-03-30 18:12 -0400;jag;Joshua "jag" Ginsberg <jag@flowtheory.net>;Piston now supports Djang
o 1.4 - thanks andialbrecht - Fixes #214ENDOFCOMMENT
296:7c90898072ce9462a6023bbec5d408ad097a362b;294:9ac392720db2 ;;piston/tests.py;2012-03-30 18:12 -0400;jag;Joshua "jag" Ginsberg <jag@flowtheory.net>;Piston now supports Django 1
.4 - thanks andialbrecht - Fixes #214ENDOFCOMMENT
```

Figure 1.4 Log file layout of Django project having Mercurial (hg) as Version Control



```
aca2d8e75e90a70a63a6695c9f61932609db212;Stefan Richter;Stefan Richter;stefanr@s5r6.in-berlin.de;stefanr@s5r6.in-berlin.de;1:2;1415722604;1415963413;drivers/firewire/core-cdev.c;
firewire: cdev: prevent kernel stack leaking into ioctl arguments
c6c748ef85c342d2726fc3c91c6a2af313af2360;Ulrik De Bie;Dmitry Torokhov;ulrik.debie-os@e2big.org;dmitry.torokhov@gmail.com;75:6;1415929686;1415929823;Documentation/input/elantech.t
xt;Input: elantech - update the documentation
```

Figure 1.5 Log file layout of Linux project having Git as Version Control

Methodology and Predictors:

The information these files had and which was of our interest is as follows –

- Total Number of Commits
- Number of Unique Commits [this were based on the unique length of the commit]
- Number of Authors
- Size of Total Number of Comments
- Size of Total Number of Unique Comments
- Delta [Number of files modified/added in one commit]

Quality Measures - Apart from this the measures we took into consideration with the data were-

- Quality of Commit
- Number of Unique Commit per Author

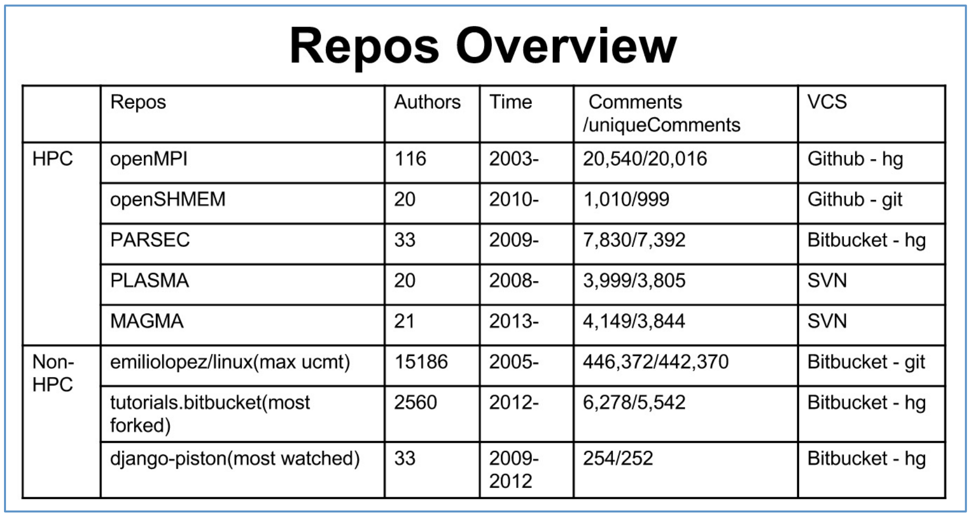
Quality Commit – It is defined as the ratio of size of the total number of unique commits to the size of the total number of commits.

$$\text{QC} = \frac{\Sigma \text{ Size of Unique Commit}}{\Sigma \text{ Size of Commit\# of Unique Commit}}$$

Number of Unique Commit per Author – It is defined as the ratio of total number of unique commit to the number of authors.

$$\text{nUC/Au} = \frac{\Sigma \text{ Total Number of Unique Commit}}{\Sigma \text{ Number of Author}}$$

Note – All these measure are calculated on year-by-year basis to know the trend in which they would work.



Repos Overview

	Repos	Authors	Time	Comments /uniqueComments	VCS
HPC	openMPI	116	2003-	20,540/20,016	Github - hg
	openSHMEM	20	2010-	1,010/999	Github - git
	PARSEC	33	2009-	7,830/7,392	Bitbucket - hg
	PLASMA	20	2008-	3,999/3,805	SVN
	MAGMA	21	2013-	4,149/3,844	SVN
Non-HPC	emiliolopez/linux(max ucmt)	15186	2005-	446,372/442,370	Bitbucket - git
	tutorials.bitbucket(most forked)	2560	2012-	6,278/5,542	Bitbucket - hg
	django-piston(most watched)	33	2009-2012	254/252	Bitbucket - hg

Figure 1.6 Overview of the Projects selected for our analysis

Results and Visualization

Commit Quality

We defined commit quality by taking into consideration Number of Commits, Number of Unique Commits and Quality Commits over the period of time for respective project. Figure 1.7 shows Commit Quality plot for all five HPC project. The sudden hike in Number of Commits at the start of each plot can be inferred as – We can say more and more files are committed at the start of the project and from there on the development start as the project gets 'stabilized' over the period of time. We can see there is no constant increase in the number of commits which somehow is coherent to our statement made. As seen for all of the plots the number of commits and number of unique commits goes hand in hand which indicates that author has taken an extra effort for each comment made in a commit, which also results in quality commit with a range of 0.90 to 1.00 which in our regard is highly appreciable. As we can see average life of the project is close to 4 years (and still running), maintaining a commit quality ratio close to 1 is robust and sturdy sign of health of the project. As most of these projects have multiple author/developers and supports complex workflows is it of great help to know a well documented history of your codebase.

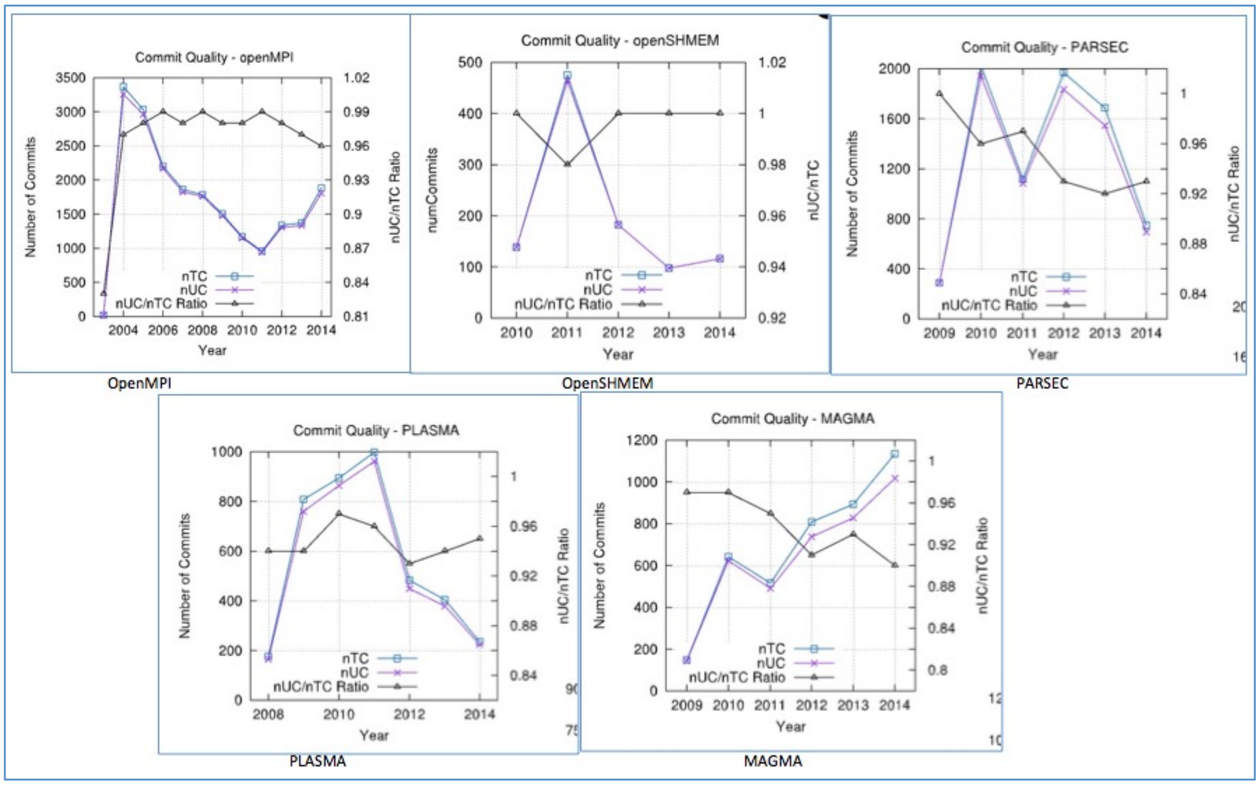


Figure 2.1 Commit Quality of HPC projects

For non-HPC projects we zeroed in on three projects from Bitbucket having maximum user commits (Linux), most forked (Bitbucket tutorial) and most watched (Django). Figure 2.2 shows the plot for commit quality for these project respectively. As seen in HPC projects there are no sharp increase at the start of the project (expect for Django); which shows a different working style of these projects. Although number of commits and number of unique again goes hand in hand but there is quite a significant variation in quality commits. The quality of commit is close to 1 for linux (most user commits) project but the same cannot be said for the Bitbucket tutorial which is having a constant decrease from its inception although it is most forked repo. It seems users/developers give very little attention to these tutorial the more they are present on Bitbucket.

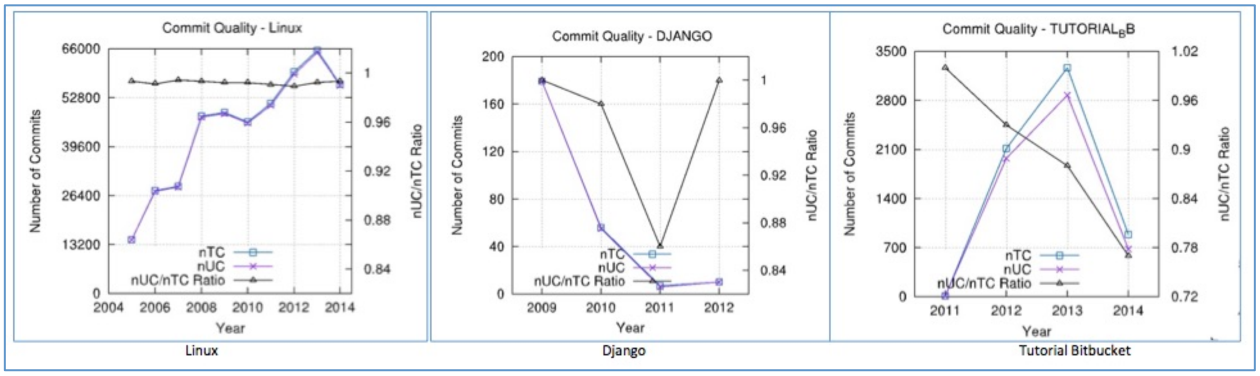


Figure 2.2 Commit Quality of Non-HPC Project

<u>Delta and Size per commit</u>

As shown in the plots (figure 2.3 and 2.4) below the size of commits in HPC projects seems to be on a higher side (around 300-1000) from which we can say that authors/developers make more effort in saying what they are doing while committing the code. On the other hand in non-HPC projects the size of commits is restricted to a range of 50-150, which is almost half of what we have in HPC. It seems authors/developers are not putting enough efforts while committing their code.

In case of delta i.e. number of files changed per commit. It seems delta is on a higher side for HPC projects, on an average 5-6 files are committed in HPC projects with highest having more than 16 (PLASMA) which shows the complexity of these projects. On the other hand files changed per commit is close to 2 in non-HPC project, looking at which one might say the ownership of the project and codebase is not very well structured. One might feel reluctant changing more files. For Bitbucket tutorial delta is almost constant, this almost seems legit as it is a tutorial for bitbucket and not many files have been changed since its inception.

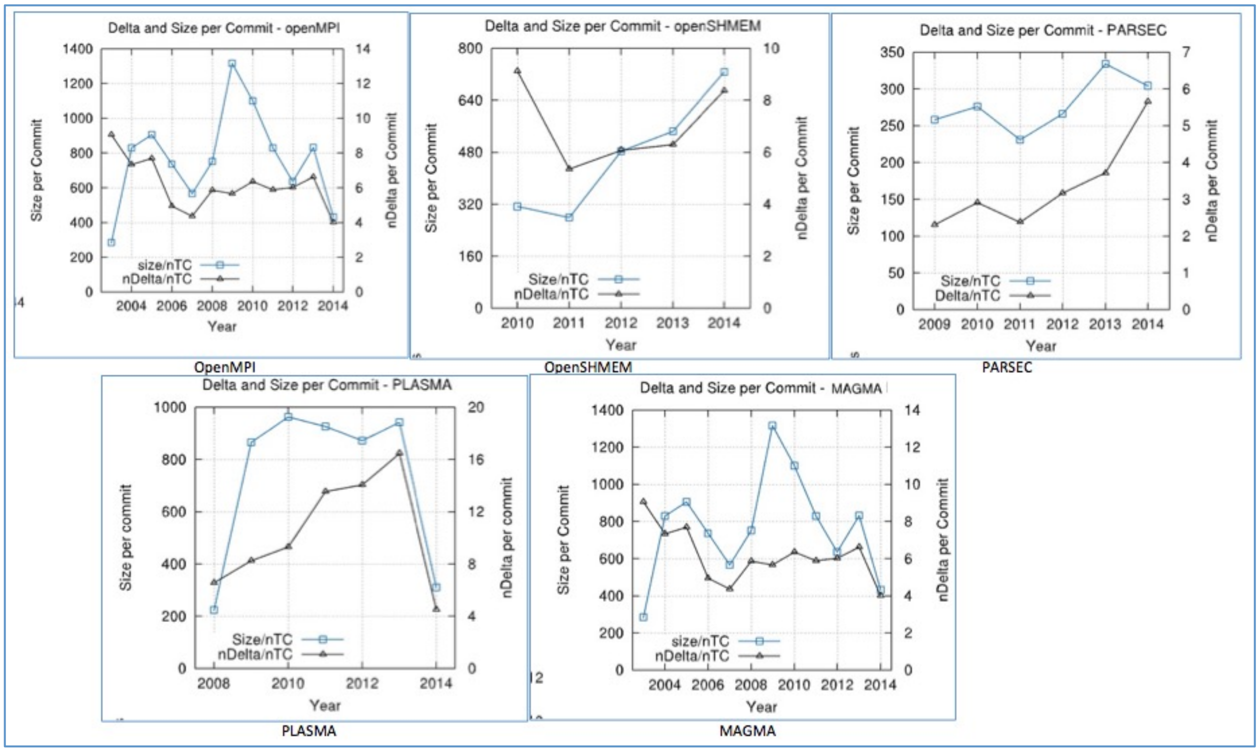


Figure 2.3 Delta and Size per Commit HPC Project

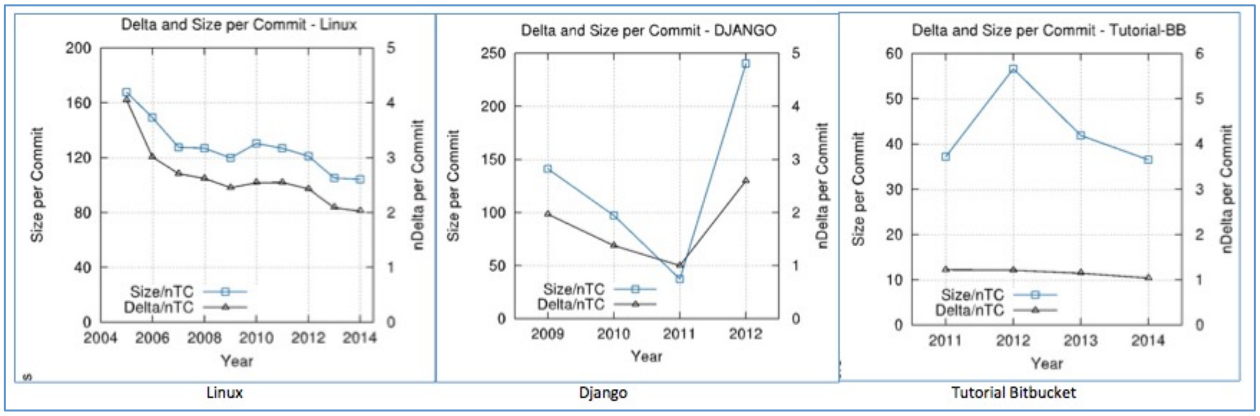


Figure 2.4 Delta and Size per Commit Non-HPC Project

Commits per Author

We were interested to know how authors for the given project work in terms of committing their changes to the codebase. We decided to measure number of unique commits per author over the tenure of the project. We wanted to know whether during a course of time, do authors fail to comply with the requirement of maintaining high quality of commit comments and how this picture shows up as each of these projects (HPC) are running on an average for more than 5 years. As we can see in figure 2.5 numbers of unique commits per author tends to decrease over the period of time. Although the actual cause of this is unknown but the hypothesis that we present here is, it might be possible that the once the author is quite familiar to the project he might shy away from putting in more effort. Also, there might be a possibility of using old comments with some changes here and there.

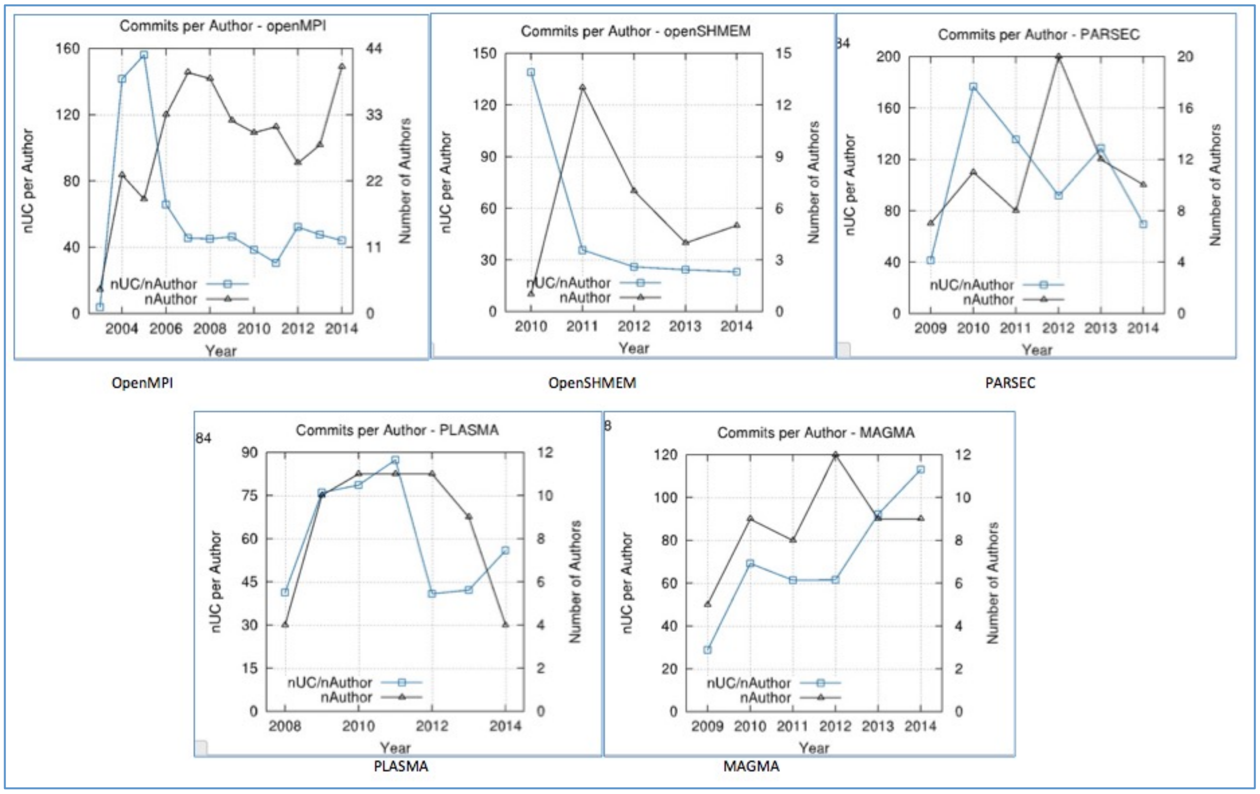


Figure 2.5 Commits per Author for HPC Project

Most of the plots in figure 2.5 are skewed to the right showing a decrease in number of unique commits per author over the tenure of the project.

For non-HPC project the hypothesis is same, number of unique comments per author tends to decrease over the period of time.

In case of number of authors it seems that HPC projects have restricted number of authors (avg. 10-15), which seems to be legit as authors of these projects have special privileges and not everyone is allowed access to these projects. The number of authors in non-HPC projects seems to be pretty high around (avg. 100-200), though as seen above the contribution of authors in development is not on par with HPC (delta is quite less for non-HPC projects).

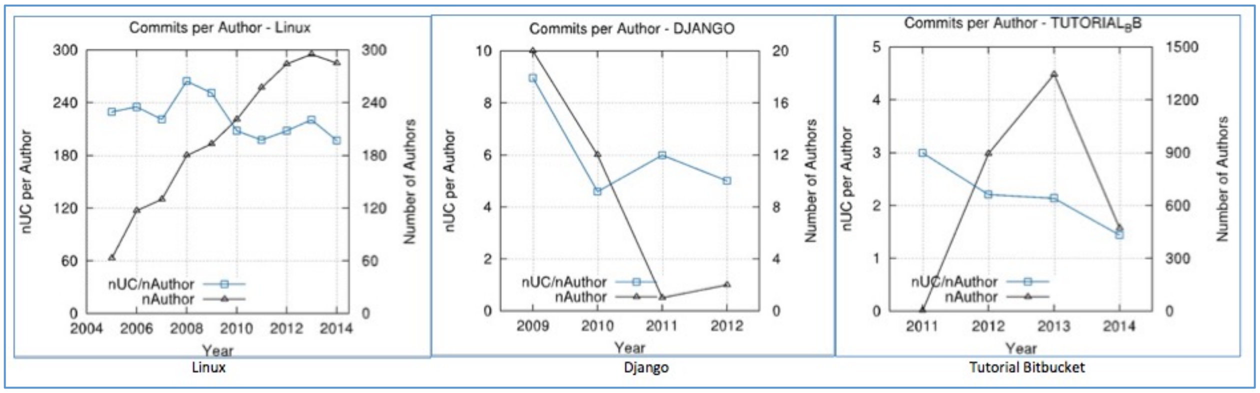


Figure 2.6 Commits per Author for Non-HPC Project

Conclusion

From our work for the duration of this project we have two different conclusion based on type of projects we studied. First, for HPC projects number of commits dropped over the tenure of the project. The contributors were quite consistent with appropriate commits. The quality of commits for HPC is high and stable (though there is a small drop over the years) and number of contributors is restricted. Second, for non-HPC projects commit quality is unstable and untrackable, number of contributors are significantly large with small number of unique commits per authors. While we are not in a position to claim that the findings will hold true in every case as software development itself is an ongoing and never ending process with many possibilities, which can throw some astonishing results. As for future attempts would be to have bring in some constitutionalized data with an outreach community to present these results with.