Scaling Up Analogy-based Software Effort Estimation: A Comparison of Multiple Hadoop Implementation Schemes

Passakorn Phannachitta[†] Jacky Keung[‡] Akito Monden[†] Kenichi Matsumoto[†]

[†]Graduate School of Information Science
Nara Institute of Science and Technology
Japan
{phannachitta-p, akito-m, matumoto}
@is.naist.jp

[‡] Department of Computer Science
City University of Hong Kong
Hong Kong SAR
Jacky.Keung@cityu.edu.hk

ABSTRACT

Analogy-based estimation (ABE) is one of the most time consuming and compute intensive method in software development effort estimation. Optimizing ABE has been a dilemma because simplifying the procedure can reduce the estimation performance, while increasing the procedure complexity with more sophisticated theory may sacrifice an advantage of the unlimited scalability for a large data input. Motivated by an emergence of cloud computing technology in software applications, in this study we present 3 different implementation schemes based on Hadoop MapReduce to optimize the ABE process across multiple computing instances in the cloud-computing environment. We experimentally compared the 3 MapReduce implementation schemes in contrast with our previously proposed GPGPU approach (named ABE-CUDA) over 8 high-performance Amazon EC2 instances. Results present that the Hadoop solution can provide more computational resources that can extend the scalability of the ABE process. We recommend adoption of 2 different Hadoop implementations (Hadoop streaming and RHadoop) for accelerating the computation specifically for compute-intensive software engineering related tasks.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Management—*Cost estimation*; D.1.3 [**Programming Techniques**]: Concurrent Programming

General Terms

Management, Measurement, Performance, Experimentation

Keywords

Software effort estimation, Analogy-based estimation, Cloud computing, MapReduce, CUDA

InnoSWDev'14 , November 16, 2014, Hong Kong, China

1. INTRODUCTION

Scaling up an effort estimation process is vital for software development, given an increasing amount of information being processed and a need of larger computing resources to facilitate the use of more sophisticated estimation technique. In the past, many important estimation parameters were usually simplified to complete the estimation process, because a sufficiently powerful computing architecture was not available [11, 20]. However, this simplification can produce different solutions and result in inaccurate estimation.

To avoid unforeseeable consequences led by the simplification, a sufficiently powerful computing architecture is clearly an interesting alternative approach. We have started discussing this approach and explored a possibility of utilizing general proposed graphic processing unit (GPGPU) technology over one of the most compute intensive methods in software development effort estimation called analogy-based estimation (ABE) [20]. In that work, we proposed ABE-CUDA algorithm. A use of CUDA technology enable us to successfully provide sufficiently computational resources that can fully optimize of the ABE process. Among many interesting features, CUDA appears to lack the ability to simply integrate more computing unit, when a problem requires more resources. Thus, scalability problem could be an inherent CUDA disadvantages. Moreover, programming in CUDA is entirely executed in a GPU device with a limited amount of internal memory. We also aware that CUDA may discourage the practitioners to extensively explore its potential use, as a development based on CUDA demands high programming expertise extending from the C language, which is usually costly to maintain, and often require lengthy development time. Thus, a software development based on CUDA seems not to be suitable for rapid development. In this work, we explore a different HPC technology that has a potential to provide unlimited scalability, at the same time with high development agility.

We explore cloud computing approach over HADOOP Map Reduce in this study. The cloud computing solution has a potential to fulfill the scalability requirement as a process can be computed in parallel across multiple computing units, where the number of machine can be simply adjusted [21]. Furthermore, a variety of interfaces to the Hadoop MapReduce programming scheme, which is flexible for different programming expertises, has a potential to fulfill a development agility requirement [22]. Different from

numerous literatures based on cloud computing, this study focuses on two main aims: (1) We propose an ABE framework based on Hadoop MapReduce as an alternative HPC solution, and (2) we examine 3 different MapReduce implementation schemes following Java native Hadoop library, Hadoop streaming, RHadoop library, and we compare with our previously proposed ABE-CUDA.

One of the highlight results was achieved by the Maxwell62 data set (i.e. the largest data set in the PROMISE repository [19]) on an experiment that uses only one master and one computing Amazon EC2 instances. Both Hadoop streaming and Java native Hadoop implementations have reduced the computing time from ABE-CUDA for 90% (i.e. from 10 hours to 40 minutes), even though; we have reported in [20] that the ABE-CUDA had already shorten the execution time from the other sequential algorithm up to 80% using commodity PC [20].

In Section 2, we outline the backgrounds and related works. Section 3 explain the 3 Hadoop MapReduce implementation schemes including a brief detail of our previously proposed ABE-CUDA. Section 4 provides detailed experimental setup and methodology, and the results are presented in Section 5. Section 6 discusses the findings, and lists some suggestion to the software engineering practitioners about Hadoop MapReduce. Finally, Section 7 concludes the paper.

2. RELATED WORKS

2.1 Cloud computing and MapReduce programming interface

US National Institute of Standards and Technology (NIST) defines cloud computing as a model for enable ubiquitous, convenience, and on-demand virtualized resources and services [18]. After the cloud-computing emergence, it has quickly gotten popular for business and IT solutions. Nowadays, well-known cloud vendors such as Amazon Web Service [2], Google Cloud Platform [8], and Cloudera [6] offer a wide range of services for business enterprises, reliable storages for big data [10], and software infrastructures including business intelligence.

MapReduce [9] is a distributed computing programming interface proposed by Google for executing their applications on a massive number of computers in their data centers. In present days, MapReduce has been well-utilized for processing and analyzing big data [10]. Its simple interfaces has attracted developer to parallelize their computational problem with ubiquitous computing environment. The main benefits of MapReduce mostly discuss simplicity and robustness. Many of well-known MapReduce implementations provide useful features for data processing such as data partitioning, data distributing, and data replication [7]. Furthermore, conventional MapReduce frameworks also integrate additional functionality to the general use of parallel computing such as. Therefore, these interesting and convenient features can make MapReduce suitable for distributed computing in the cloud [7].

In an abstract view of the programming model, MapReduce has two main functions; Map and Reduce. The master node begins an execution in MapReduce by scattering the program instruction to the compute node using the Map function, where the input data are already located. When processing with MapReduce, a task communicates each other with Key-Value pair (<K,V>) protocol. Program instruction inside the Map function has to transform the input data into <K,V>, process them, and emit the computing outcome as an intermediate <K,V>. The reduce function will receive the intermediate <K,V> from the Map function, group chunks of <K,V> into <K,[V]> by keys, and processes the <K,[V]> to produce the final output <K,V>, which will be the output of the MapReduce program.

2.2 Analogy-based estimation

Analogy-based Estimation (ABE) has been successfully become a well-known software effort estimation procedure. Its essential hypothesis mimics the basic reasoning process of human being, such that "software projects with similar characteristics should require similar efforts to produce". While the key hypothesis of ABE may sound straightforward, the estimation performance based on it can challenge many other sophisticate model-based estimation approaches [16].

Estimating a development effort using ABE heavily relies on parameter optimization during the process. As the method is to find the most similar software project case and adapt it to the query, examples of the important parameters are: (1) a appropriate method to discuss the similarity between project pairs, (2) exact number of similar software project given by a query, and (3) important software project features to discuss similarity. To reliably determine these important parameters, the ABE process requires an exhaustive search for not only a long list of possible parameters and instances under each dimension, but also the combinations across them. Therefore, ABE is an interesting problem to explore a possibility to facilitate high-performance computing (HPC) technology for software engineering process.

3. ABE ALGORITHM AND MULTIPLE HPC IMPLEMENTATIONS FOR ABE

Algorithm 1 presents a pseudocode of the simplified sequential version of the ABE optimization process scoping from reading a software project input, until achieving the optimized parameter set containing the best software project features and best number of analogies (k) that provide the minimum error.

Algorithm 1 reads and assigns variables between Line 1 and 5. Table D holds the input data set, where column vectors indicate feature variables, and row vectors indicate project cases. Vector f represent feature sets, vector t holds the actual effort of all the project case, vector R initializes a container for the result, and $MinimumError$ is initialized as 0 in the preprocess. The outer for-loop between line 6 and 21 iterates all feature subset combinations to searching for the most influential one. Dissimilarity between each pair of project case is constructed in Line 7, and store in Distance matrix. The matrix will then reordered in Line 8 to represent look up table. A for-loop between Line 9 and 20 looks up for the best number of analogies (k), and its inner loop between line 11 and 15 examines error between the actual effort and the estimated effort. Once an error for particular feature set and k is produce, the algorithm will check if it is the global optimum in a if-condition between line 16 and 19. The output of Algorithm 1 will present minimum error, the most influential software project feature set and the best number of analogies.

Algorithm 1 ABE optimization process

1: $D \leftarrow$ read the input data set into table format
2: $f \leftarrow$ define the feature sets
3: $t \leftarrow$ read the actual effort from input data set
4: $R \leftarrow (\infty, nil, nil)$
5: $MinimumError \leftarrow 0$
6: **for all** f_i **in** combination(f) **do**
7: construct $DistanceMatrix$ from D and f
8: $SortedMatrix \leftarrow$ sort $DistanceMatrix$ by case
9: **for** $k = 1 \rightarrow \#case(D)$ **do**
10: $error \leftarrow 0$
11: **for** $i = 0 \rightarrow \#f$ **do**
12: $Actual_i \leftarrow t_i$
13: $Predict_i \leftarrow Average(t_{Rank(SortedMatrix_{0..k})})$
14: $error \leftarrow error + \frac{1}{\#case(D)} \times \frac{|Actual_i - Predicted_i|}{Actual_i}$
15: **end for**
16: **if** $error < MinimumError$ **then**
17: $MinimumError \leftarrow error$
18: $R \leftarrow (MinimumError, f_i, k)$
19: **end if**
20: **end for**
21: **end for**

3.1 ABE-CUDA

We proposed ABE-CUDA in [20] to enable the ABE framework to estimate large data set. ABE-CUDA tailors ABE to a HPC framework called CUDA compute architecture to accelerate massive amount of computation using the power of graphic processing unit (GPU). CUDA was chosen over other HPC solution in that work, because it was proved as an efficient computing paradigm that can allocate massive amount of computing power even a developed task is deployed on a single machine [5].

ABE-CUDA implements 4 GPUs kernel functions from the two main components of ABE (i.e. Distance matrix construction and Parameter optimization). Distance matrix construction computes distance (similarity) between all pairs of project cases for each particular feature subset and a number of analogues (k), and Parameter optimization module searches for the parameter set that produces the minimum estimation error from the input data set.

3.2 ABE-Hadoop

To explore more spectrum of the HPC solutions over the ABE, we present and compare 3 more HPC solutions base on Hadoop MapReduce with ABE-CUDA in this study. As all the 3 HPC solutions tailor the Hadoop MapReduce, let we call them ABE-Hadoop in short. ABE-Hadoop modifies Algorithm 1 from the loop at Line 6 as follows:

Algorithm 2 computes the error for a given feature subset and an observing k value following the original Algorithm 1. The outputs of the Map function are a collection of intermediate $\langle keys, values \rangle$ pairs, which are consisted of k pairs of $\langle error, (k, f) \rangle$. The Reduce function takes the intermediate $\langle keys, values \rangle$ form the Map function and search for the pairs that contains the minimum error. The output of Algorithm 2 is the feature subset in corresponding to the number of analogies, and the error value the produce them minimum error. Figure 1 summarizes the ABE-Hadoop process flow.

Algorithm 2 ABE-Hadoop optimization process

1: **function** MAP($\langle key, value \rangle$)
2: $f \leftarrow key$
3: construct $DistanceMatrix$ from D and f
4: $SortedMatrix \leftarrow$ sort $DistanceMatrix$ by case
5: **for** $k = 1 \rightarrow \#case(D)$ **do**
6: $error \leftarrow 0$
7: **for** $i = 0 \rightarrow \#f$ **do**
8: $Actual_i \leftarrow t_i$
9: $Predict_i \leftarrow Average(t_{Rank(SortedMatrix_{0..k})})$
10: $error \leftarrow error + \frac{1}{\#case(D)} \times \frac{|Actual_i - Predicted_i|}{Actual_i}$
11: **end for**
12: Emit $\langle error, (k, f) \rangle$
13: **end for**
14: **end function**

1: **function** REDUCE($\langle keys, values \rangle$)
2: $\langle MinError, (Best_k, Set) \rangle \leftarrow \langle \infty, (\infty, nil) \rangle$
3: **for all** $\langle key \in keys \rangle$ **do**
4: **if** $key < minError$ **then**
5: $\langle MinError, (Best_k, Set) \rangle \leftarrow \langle key, values \rangle$
6: **end if**
7: **end for**
8: Emit $\langle MinError, (Best_k, Set) \rangle$
9: **end function**

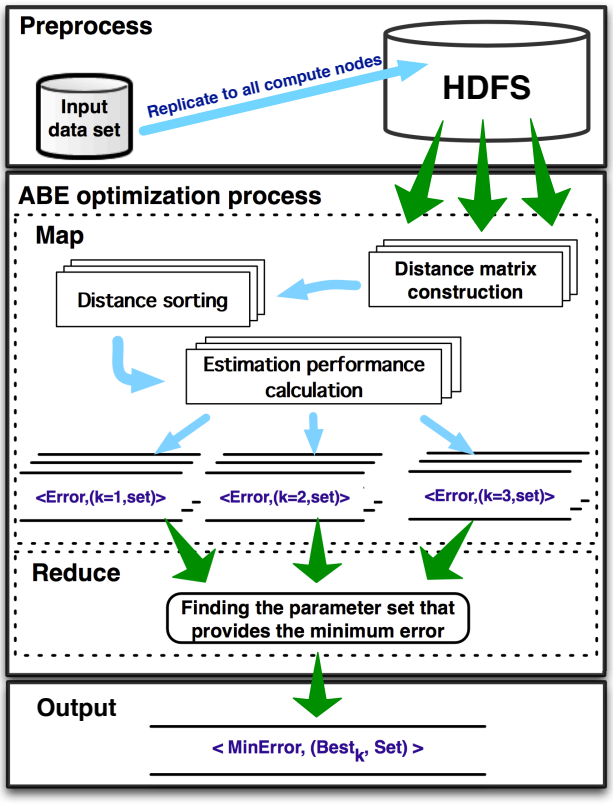

Figure 1: A graphical overview of ABE-Hadoop

3.2.1 ABE-JavaNativeHadoop

ABE-JavaNativeHadoop is an implementation of ABE framework we have reimplemented all the ABE-CUDA kernels into Java language following MapReduce Java library provided by Hadoop. The Map function computes both Distance and Optimization functions over all the combinations of ABE parameters. The Reduce function collects the estimation performance given by particular ABE parameter, and reduces all the possible combination into the best one that produces the minimum error.

To the best of our knowledge, the Hadoop platform and framework have been developed only in Java programming language by the time of this writing. Therefore, if a program is reimplemented following the Hadoop-provided Java MapReduce library, minimum additional cost to execute the problem will allow the program to be inherently and fully optimized. However, the additional cost would be more spent on the process to reimplement a problem into Java that will require an extensive effort with Java programming expertise as some parallel programming skills.

3.2.2 ABE-HadoopStreaming

In the ABE-HadoopStreaming implementation scheme, we deployed the sequential version of ABE-CUDA which entirely processes all the functions using CPU into Hadoop. We firstly divided a the data to examine into smaller chunk (e.g. there are 2^f feature combinations divided by number of computing cores in each chunk). Then, we used a Hadoop streaming command (i.e. $HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar) to tell all the computing instances to distributed execute the sequential ABE program in parallel. The outputs from all the Map function instances are an estimation error for a given feature subset. To find the minimum error and the best feature subset, we implemented a Reduce function using simple BASH script language, which does the same task as that of ABE-JavaNativeHadoop. The Reduce function collects the estimation performance given by particular ABE parameters from multiple machines, and reduces the intermediate result into one that produces the minimum error.

Hadoop streaming is an interface that fully enhances programming adaptability and extensibility to Hadoop. The interface allows any executable, script, or even a single command to be executed across multiple machines. Hadoop streaming is flexible and programming language independent, so that it allows a developer who has a limited expertise of Java programming language or parallel programming skill to acquire the power of HPC for their application. However, computing adaptability is inevitably sacrifice the performance. Comparing to the JavaNnativeHadoop, Hadoop streaming appears to lack the accelerated performance.

3.2.3 ABE-RHadoop

In ABE-RHadoop, we reimplemented all the 4 CUDA kernels using R script language. This is the simplest version we have implemented among the 3 Hadoop and 1 CUDA implementations. It requires minimum expertise in both programming and parallel programming skills. For the process flow, we entirely setup all the ABE parameters to be examines as well as the data dividing process in the R framework. Thus, the entire source codes for this version were implemented in the R language.

RHadoop is a collection of R statistical computing and data mining library provided by RevolutionAnalytics [1]. RHadoop allows users to manage and analyze big data using a wide range of R packages in parallel with Hadoop. A parallel program can be implemented in a R, which is a high-level script language. With an ability to facilitate higher computational power while the development overhead is remaining low, the RHadoop is an interesting solution that eases the development agility.

4. EXPERIMENTAL SETUP

4.1 Controlled Experiments

The conditions developed in our evaluations are listed as follows:

1) The estimated performances were optimized over the parameters that we had set in ABE-CUDA [20]. From the influential parameters of ABE [3], we chose to optimize a search for (1) the feature variable combination that produces the minimum error, (2) the number of similar projects that produces the minimum error. We (3) normalized the value using $max - min$ for all the continuous variable. We (4) selected the Euclidean distance family as a similarity measure, and (5) we chose unweight average as solution adaptation method.

For training and testing, we chose leave-one-out validation technique (a.k.a. Jack-knifing) as a sampling method. This is another component that demands computational power. Leave-one-out separates one software project case as a test instance of the model that being built from all the remain cases, and it repeats this process for all the cases [13]. It is a technique that is commonly selected over the other procedures in ABE literatures [12, 14], because of lower bias and higher variance generated.

2) We controlled the experiments under condition (1) and replicated the experiments over ABE-CUDA, ABE-JavaNative Hadoop, ABE-HadoopStreaming, and ABE-RHadoop. We recorded the total computing time for every single run across different number of computing instances to present the accelerated performance, and we further analyzed the results to observe scalability.

4.2 Testbed environments

We deployed the experiments in two different Amazon EC2 instances. We chose the GPU G2 instance for ABE-CUDA, because the maximum throughput from from GPUs can be expected with this instance type. However, we ware aware that the result could not be optimum if we run the process that utilized only CPU on the GPU-optimized instance. Therefore, we chose another CPU-optimized instance for the 3 Hadoop implementations. Table 1 describes the hardware, software, and configuration per one computing instance.

We deploy ABE-CUDA in a single G2 instance, and we set up 2, 4, and 8 C3 instances for the rest 3 implementations based on Hadoop to observe the computing efficiency and scalability. In all the Hadoop setups, we assigned an instance as a master node (i.e. Namenode and Jobtracker), and leave the rest as compute node (i.e. Datanodes and TaskTrackers). For example, a 4-instance setup has 1 master and 3 compute instances. We were limited the access to 8 C3 computing instances, because the 32-cores C3 instance is the recently most powerful computing instance, and it require a special approval to get an accessibility to more number of instances.

Table 1: The 2 testbed environments from Amazon EC2

	Compute-optimized c3.8xlarge	GPU g2.2xlarge
Processor	Intel Xeon E5-2680v2	Intel Xeon E5-2670
#Cores	32	8
DRAM	60 GiB	15 GiB
#GPU cores	N/A	1,536
VRAM	N/A	4GB
Storage	30 GiB SSD	30 GiB SSD
Network	10 Gigabit	N/A
OS	64-bit Linux 3.1 based Amazon AMI	
Hadoop Library	Apache Hadoop 1.2.1	

Nonetheless, we will present in the next section that only 8 C3 computing instances are considerably powerful enough to handle any ABE task over the recently available data sets.

4.3 Data sets used in the case study

We selected 2 large data sets from the PROMISE software engineering repository [19] as our experimental data sets. The PROMISE repository is where empirical software engineering datasets were collected and made available for software engineering researchers and practitioners to tackle challenges in software engineering problems, such as bug prediction and software development effort estimation. The two data sets themselves seem not to be large in physical size (i.e. can be seen as a table with a less than 100 rows and columns), however; an intensive manipulation over their entries makes them heavily compute intensive. Table 2 summarizes the properties of the two data sets.

Table 2: The data sets

Data set	Nasa93	Maxwell62
# Projects	93	62
# Features	23	26
Description	Nasa software projects [4]	Projects from commercial banks in Finland [17]

As the ABE optimization process examines all the features subset combinations, there will be at least 2^{23} and 2^{26} operations computed over the Nasa93 and Maxwell62 data sets respectively. In the next section, we will present an empirical benchmark comparing the performance based on the 4 different implementations on GPU and Hadoop using the Amazon EC2 computing environment.

5. RESULTS

Figure 2 and Figure 3 present the total execution times (i.e. wall-clock time), when repeating the ABE optimization process over the Nasa93 and Maxwell62 data sets.

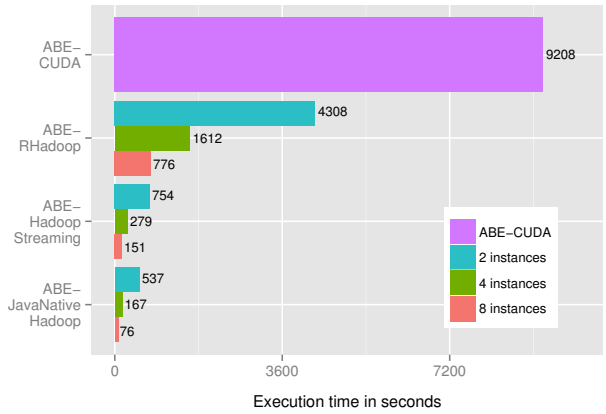

Figure 2: Comparing 4 ABE implementations over the Nasa93 data set

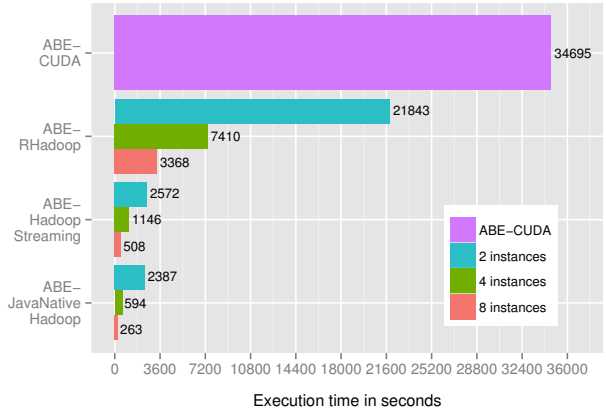

Figure 3: Comparing 4 ABE implementations over the Maxwell62 data set

While our previous study has presented that the ABE-CUDA could shorten the execution time from the other sequential implementation that used only CPU up to 80% in a commodity PC [20], all the alternative Hadoop implementations in this study have displayed further impressive improvement over both Nasa93 and Maxwell62 data sets. From the execution time over the Nasa93 data set as presented in Figure 2, ABE-JavaNativeHadoop was the fastest following by ABE-HadoopStreaming, ABE-RHadoop, and ABE-CUDA. The execution time in seconds showing on the x-axis presents that even an execution using only one compute node (2 instances), the slowest Hadoop implementation still notably faster than that of the ABE-CUDA. The computing instance equipped with a 32-cores CPU made the ABE-RHadoop spent less 50% the computing time than the ABE-CUDA. Furthermore, the ABE-HadoopStreaming and the ABE-Java NativeHadoop were outstandingly faster, they had reduced the execution time from the ABE-CUDA for over 90% using the same single compute node setup.

The Maxwell62 data set has presented similar result from that of the Nasa93. Figure 3 presents a similar trend to Figure 2 but in a different scale. Maxwell62 has 3 more feature variables than the Nasa93 that will have at least $(2^{26}) - (2^{23}) = 58,720,256$ more operations to compute. From the

Table 3: Summary of fundatemttal source code metrics over 4 implementation schemes

	ABE-CUDA	ABE-JavaNativeHadoop	ABE-HadoopStreaming	ABE-RHadoop
Language	C++, CUDA	Java	C++	R
LOC	2,095	1228	1,674	471
#Files	40	18	35	17
#Functions	119	82	90	21

result, executing the ABE process using RHadoop over 2 instances still requires lengthly time to complete (i.e. approximately 6 hours), whereas the ABE-HadoopStreaming and the ABE-JavaNativeHadoop require less than an hour. Thus, the Hadoop implementations over powerful cloud- computing instances could shorten the computing time from the hours timeframe into a few minutes.

The computing time has been commonly reduced by approximately 50% when increasing the number of computing instances twice. In Nasa93, increasing from 2 to 4 instances, ABE-Rhadoop reduced the execution time by 52%, ABE-HadoopStreaming reduced the execution time by 63%, and ABE-JavaNativeHadoop reduced the execution time by 69%. Likewise, when increasing from 4 to 8 instance, the results still retain to approximately the same ratios. When the hardware resources were doubled, all the 3 Hadoop-based implementations have reported more than 2 times speed-up. The reason is we always assigned one computing instance as the master node and let it take care only the operation and communication tasks, thus; the computing instances that actually computed the task were not doubled as the number in Figure 2 and Figure 3 had indicated. However, we also investigated for other possibilities, and we believe that the execution time spent on the Reduce function also have an influence to the circumstance. Increasing number of computing instances can reduce the total stall time that the Reduce functions have to wait for the outputs from the Map functions, because the data sets had been divided into more number of smaller chunks, and each chunk will require less computing time to complete.

When executing the ABE-HadoopStreaming and the ABE-JavaNativeHadoop over 2 computing instances, the two implementations have spent almost equal computing time. Therefore, when the input data are not massively large or the complexity of their application does not growth in exponential, we suggest Hadoop Streaming would be a better solution, This is because Hadoop streaming will require much less effort to migrate the task. Also we recommend to allocate more budget for more computing instances rather than preparing to spend much over the migration process. Note that the set up time for initiating a Hadoop MapReduce framework in typical cloud is considerably low. In this study, we spent less than 20 minutes to start 8 computing instances in the Amazon EC2.

5.1 Development agility

To discuss the development agility, we computed several fundamental software matrices over our implemented source codes. We used a source-code analysis toolset named Understand [15] for C++ and Java code, but we had to analyzed the R source code manually, because; a tool set that can analyze R source code is not existing. Table 3 presents the source-code metrics computed over the 4 implementations of ABE.

In a common real-world scenario, developers will initially have an application implemented in sequential such as we have 1,674 LOC sequential program (i.e. in Table 3-Hadoop Streaming). To make it run in HPC, HadoopStreaming can be the most attractive choice as long as we do not need much more effort to migrate it. Next, CUDA is also a good option as this program was written in C++. In such case, we could reuse all the original source code such that in our example we had to implement more 421 LOC (about 25%) to enable the original source file executable in GPU. In contrast, we had to reimplement the whole framework in Java or R to migrate that application in either JavaNativeHidoop or Rhadoop. From our examples, our reimplemented source code in Java and R consume 1,228 LOC (about 70%) and 471 (28%) LOC respectively.

There are cases that Java and RHadoop seem to be the better choices. JavaNativeHadoop will be a suitable choice in case the source program was already well-written in Java or the developers have expertise in Java programming language as well as parallel programming skill. On the other hand, we will choose the RHadoop in case we are going to start developing a new software project from scratch. In our R implementation, LOC, number of files, and number of functions, are entirely less than the value recorded from all the other implementations.

6. DISCUSSIONS

6.1 ABE over Hadoop MapReduce

This study has presented that a HPC solution over Hadoop MapReduce is not only suitable for big data processing [10], but it can also provide a massive amount of computation power for compute intensive tasks. We have explored a fully-utilized Hadoop MapReduce framework to transparently send the instruction set of the ABE process to multiple 32-cores CPU computing instances and let the process computed concurrently. From the results, all the implementations based on Hadoop MapReduce have achieved impressive accelerated performance. An implementation based on Java library provided by Hadoop was the fastest execution. Because an implementation in Java is in the same programming language as the native Hadoop platform, all the functions require the least overhead. Interesting findings in which different from our presumption is the execution time over Hadoop streaming is not much slower than the Java Hadoop, although; it supposes to include much more overhead in order to allow any program executed in the Hadoop platform. From this finding, we would encourage software engineering practitioners in any domain to consider Hadoop for their problem solving or applications. We suggest a combination

Table 4: Summary of and suggestion towards the findings in this study

	ABE-CUDA	ABE-JavaNativeHadoop	ABE-HadoopStreaming	ABE-RHadoop
Effort required for development	Very high	Moderate	Depends on programming language	Very low
Effort required for migration	Moderate - source is in C++ Very high - source is in other language	Low - source is in Java High - source is in other language	Low	Very low - source is in R Low - source is in other language
Require high expertise on	C++, CUDA, HPC	Java, HPC	Any programming language	-
Require optional expertise on	-	-	HPC	R programming language, HPC
Scalability	Low	Very high	Very high	High
Suitable for	Executing a HPC task in a local machine	Migrating a task that is well-written in Java to the Cloud	Migrating any task to the Cloud	Start developing a HPC task from scratch and plan to deploy it in the cloud

between Hadoop streaming and high performance Amazon EC2 as an interesting solution to any software engineering problem that requires a massive amount of computational power. Migrating a program into Hadoop using Hadoop streaming is almost effort-less, as the Hadoop streaming allows any executable file to be the Map function of the MadReduce. Thus, neither additional expertise in parallel programming nor advance Java programming is required.

The RHadoop is different from the two other ABE-Hadoops. Although it is the slowest among the 3 Hadoop implementation schemes, we would still recommend it for any development task that concerns development agility. To develop a task from scratch or analyzing data with a wide-range of complex mathematical functions, implementing with R will be clearly more convenience than other solutions. In the past, computing speed was one of the biggest challenge of utilizing R for a real-world problem. From our results, RHadoop can accelerate sufficient performance to make a programming based on R no longer lack the accelerated performance. Therefore, we recommend a combination of RHadoop and high-performance Amazon EC2 instances for to ease a challenge in compute-intensive problem.

Recently, a commodity PCs equipped with a 32-cores CPU is not available in the consumer market, thus; the performance presented from our results are only valid on the cloud. Therefore, ABE-CUDA is remaining the best option for estimating the effort in a local machine, even through; ABE-CUDA requires very high effort to develop. We discuss the accelerated performance and present the scenario to access the rank of HPC solution based on GPU and Hadoop MapReduce, and Table 4 summarizes our findings and suggestions from this study.

6.2 Threats to validity

Internal validity: We have implied the scalability from our results that the implementation based on Hadoop MapReduce can scale larger than the CUDA implantation. At least we can present that computing the ABE process over a large data set, which took longer than a day in the past, could be finished on a virtual computing cluster consisting of 8 computing instances in a few minutes. However; we are unable to obviously conclude the scalability because a sufficiently large and accessible software engineering data set does not exists available data set. Possible future work from this point is to undertake the experiments with large artificial data sets.

External validity: The results on development agility may not able to actually generalize the characteristic of all the 4 different implementations in a full spectrum, however; it can guide some helpful insights for practitioners to roughly decide their suitable HPC implementation procedure. To fully access the rank among these procedures, future works are required to repeat the experiments over many different subject developers, and observe their experience.

7. CONCLUSION AND FUTURE WORK

The highlighted result from this study is that the cloud computing solution is not just perfectly fit for handling big data tasks [10], but it can also present an impressive outlook for massively compute-intensive applications. In this study we have put an emphasize to scale up an effort estimation based on analogy (ABE) method using Hadoop MapReduce, as an interface to the cloud computing environment. ABE is an example of software engineering process in which the input data is not to physically large, but massive manipulation over the data has introduced an insufficient computational resource problem in the past. In this study, we have presented 3 implementation schemes based on Hadoop MapReduce for the ABE, and compared all of them in contract with our previously proposed ABE-CUDA [20] framework, on the Amazon EC2 cloud service platform. All the 3 ABE implementations over MapReduce have outperformed ABE-CUDA on the high performance computing instances equipped with 32-cores CPU. A Hadoop implementation following RHadoop library spent 50% less computing time than the ABE-CUDA, because an exceptionally large amount of computing cores supported by the EC2 were fully utilized. Furthermore, the other two implementations following Java

native Hadoop library and another implementation that directly deploys an executable file through Hadoop by streaming, could outstandingly reduce the execution time from the ABE-CUDA for over 90%.

At higher number of computing instances, the difference in computing performance becomes very small when comparing between the Java native Hadoop and the Hadoop streaming version, even though; developing a task using Hadoop streaming is much more simple to undertake. We suggest software engineering practitioners to reduce the migration cost by selecting the Hadoop streaming whenever their original program was not developed in Java language, and rather spent the reduced cost to fund on more computing instances. We have performed further experiments over 8 computing instances to observe scalability. At this number of instances, all the 3 implementation schemes have further reduced the computing time. Over Nasa93, one of the largest available effort estimation data set, the total execution time was decreased to approximately 1 minute, though; the same task could take up to 20 hours in a sequential run using commodity PC [20].

Compute-intensive software engineering experiments can be implemented using readily available computing resources from the public cloud, our example effort estimation task illustrated in the study has provided an important insights to utilize these resources to facilitate and accelerate the development of empirical software engineering research.

Acknowledgements

This research has been conducted as a part of "Research Initiative on Advanced Software Engineering in 2013" supported by Software Reliability Enhancement Center (SEC), Information Technology Promotion Agency Japan (IPA), Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C) (26330086), the City University of Hong Kong research fund (Project No. 7200354, 7003032), and the Global Initiatives Program for Promoting Overseas Collaborative Research Toward Graduate Education in Biological Sciences, Nano Science, and Information Technology, Nara Institute of Science and Technology. And all the experimental testbeds have been sponsored by the AWS in Education Research Grant award.

8. REFERENCES

[1] J. Adler. *R in a Nutshell, 2nd Edition*. O'Reilly Media, 2012.

[2] Amazon Web Service. http://aws.amazon.com/, 2014.

[3] M. Azzeh. A replicated assessment and comparison of adaptation techniques for analogy-based effort estimation. *Empirical Softw. Engg.*, 17(1-2):90–127, 2012.

[4] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1981.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron. A performance study of general-purpose applications on graphics processors using cuda. *J. Parallel Distrib. Comput.*, 68(10):1370–1380, 2008.

[6] Cloudera. http://www.cloudera.com/content/cloudera/en/home.html, 2014.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1), 2008.

[8] Google cloud platform. https://cloud.google.com/, 2014.

[9] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox. Mapreduce in the clouds for science. In *Proceeding of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 565 –572, 2010.

[10] IBM, P. Zikopoulos, and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[11] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery. Analogy-x: Providing statistical inference to analogy-based software cost estimation. *IEEE Trans. Softw. Eng.*, 34(4):471–484, 2008.

[12] E. Kocaguneli, T. Menzies, and J. Keung. On the value of ensemble effort estimation. *IEEE Trans. Softw. Eng.*, 38(6):1403–1416, 2012.

[13] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1143, 1995.

[14] M. V. Kosti, N. Mittas, and L. Angelis. Alternative methods using similarities in software effort estimation. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, pages 59–68, 2012.

[15] R. Lincke, J. Lundberg, and W. Löwe. Comparing software metrics tools. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 131–142, 2008.

[16] C. Mair and M. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *Proceedings of 2005 International Symposium on Empirical Software Engineering (ISESE)*, page 10, 2005.

[17] K. Maxwell. *Applied Statistics for Software Managers*. Englewood Cliffs, NJ. Prentice-Hall, 2002.

[18] P. Mell and T. Grance. The nist definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

[19] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data. http://promisedata.googlecode.com, Jun 2012.

[20] P. Phannachitta, J. Keung, and K. Matsumoto. An empirical experiment on analogy-based software cost estimation with cuda framework. In *Proceedings of the 22nd Australian Conference on Software Engineering*, pages 165–174, 2013.

[21] W. Shang, B. Adams, and A. E. Hassan. An experience report on scaling tools for mining software repositories using mapreduce. In *Proceedings of the IEEE/ACM international conference on Automated software engineering (ASE)*, pages 275–284, 2010.

[22] Symantec.cloud. Weathering the storm : Considerations for organizations wanting to move services to the cloud. White Paper, 2011.