A Performance Prediction Scheme for Computation-Intensive Applications on Cloud

Hongli Zhang[1], Panpan Li[1], Zhigang Zhou[1], Xiaojiang Du[2] and Weizhe Zhang[1]
[1]School of Computer Science and Technology, Harbin Institute of Technology, Harbin, 150001, China
Email: zhanghongli@hit.edu.cn, {lipan, zhouzhigang, zwz}@pact518.hit.edu.cn
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, PA, 19122, USA
Email: dux@temple.edu

***Abstract*—As cloud computing services are gaining popularity, many organizations are considering migrating their large-scale computing applications to cloud. Different cloud service providers (CSPs) may have different computing platforms and billing methods. Most cloud customers don't know which CSP is more suitable for their applications and how much computing resource should be purchased. To address this issue, in this paper, we present a performance prediction scheme that allows a cloud customer to accurately predict computing resource (e.g., running time) for an application. The proposed scheme identifies application's control flow and scaling blocks, constructs a miniature version program to run in local machines, and then replays it in cloud to get the performance ratio between local and cloud. Our real-network experiments show that the scheme can achieve high prediction accuracy with low overhead.**

***Index Terms*—Cloud computing; performance prediction; cost**

I. INTRODUCTION

Cloud computing has gained tremendous popularity in recent years. A number of large international IT companies have developed their own cloud platform, e.g., Google App Engine and Amazon's EC2 and S3 [1, 5]. Many individuals and organizations outsource their computation and storage to cloud. Cloud computing creates a new way for IT operation and management. By using resources from CSPs, cloud customers can save the cost of purchasing hardware and software systems, hiring IT personnel, and system operation and maintenance expenses.

However, given some many different cloud platforms, choosing the best cloud platform is not a trivial problem. Different CSPs may offer different service models, such as platform as a service (PaaS, e.g., Google App Engine [5]), and infrastructure as a service (IaaS, e.g., Amazon EC2 [1]). In addition, different CSPs offer different options in pricing, performance, and feature set. For cloud customers, due to lack of IT expertise, they don't have a good idea on how many cloud resources (e.g., store and computing power) should be purchased. Answering this question may benefit both cloud customers and CSPs. For cloud customers, the answer can help them choose the right CSP and pay the right amount of cloud resources for their IT tasks. For CSPs, the answer may help them set up fine-grained charge standard [6].

In this paper, we propose a performance prediction framework that can accurately predict the cost of customer applications without migrating them from local to cloud. In our framework, first, we find the performance scaling blocks in customer's applications, and then we create a miniature version of the program. Our prediction framework traces the running of the miniature version program using a lightweight trace engine, and then replays it on cloud to get the performance scale ratio between local and cloud. The replayer hides complex program logics and platform details, while emulating the performance equal scaling blocks of the real program on cloud. We implement the framework and deploy it on Eucalyptus [15]. Taking HPL (High Performance Linpack) benchmark as our test case [2], the evaluation shows that the proposed framework can accurately predict the time cost of customer applications with low overhead.

Our contributions are summarized as follows.

- To the best of our knowledge, this is the first work that studies the problem of performance prediction for computation-intensive applications on cloud.
- We present a performance prediction framework, which can accurately predict the time cost of user applications on cloud. Our method doesn't need actual migration.
- We implement the framework on cloud and the HPL benchmark tests show that our scheme can accurately predict the time cost with low overhead.

The rest of the paper is organized as follows. In Section II, we describe the problem. And then focusing on computation-intensive application, we provide a performance prediction framework and give the performance keys model as criterion of evaluating performance cost in Section III. Section IV and Section V describe the performance model in detail. We use experiences to validate the accuracy and lightweight-injected of the framework in Section VI. Finally, we overview the related work and give the conclusion of the whole paper in Section VII and Section VIII, respectively.

II. THE PROBLEM STATEMENT

Our goal is to predict the cost (e.g., running time) of computation-intensive user applications when they run in an IaaS cloud platform, such as Amazon EC2 [1]. We focus on computation-intensive applications because they are one of the most popular types of computing applications running on the cloud. Nowadays, many customers choose to run their applications on cloud platforms because it saves money. A user just leases computing and storage resources with the pay-as-you-go manner [3, 4, 6].

However, there are some challenges: (1) Cloud computing environments vary widely cross platforms, even for the same application, the running time may be very different on different cloud platforms. Some existing research achievements look at the problem from the application layer: determining the running time only based on the number of loops, which is not accurate. (2) Computation-intensive applications are diverse, and the running time of different applications may vary a lot. Even for the same application, the running time is nonlinear of the input data size. (3) Many factors affect computing cost, and the OS could shield the details. Therefore, it's difficult to measure influence of each factor in an application [9, 14].

In addition, to ensure accurate prediction of application performance by not introducing heavy additional overhead, the injected code to our framework should be lightweight.

III. THE SYSTEM ARCHITECTURE

Motivated by the aforementioned challenges and design rationales, we design a performance prediction framework consisting of three components: the decomposition engine, the control flow extractor, and the data flow extractor. Figure 1 illustrates the performance prediction framework.

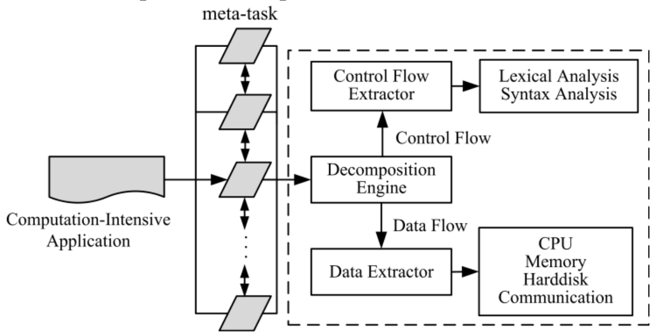

Figure 1. Architecture of the performance prediction framework

The function of the decomposition engine is simple. It translates a computation-intensive application to a simple, formally specified intermediate language and then provides a set of core utilities for common static analysis on the intermediate language. It is like a distributor that divides a computation-intensive application into two parts: control flow and data flow.

The control flow extractor is a static analyzer for the entire system, which enables the entire-system monitoring and finds the performance equal scaling blocks based on lexical analysis and syntax analysis. The data flow extractor is a dynamic analyzer, and it compares the time cost of each aspect between local and cloud. First, the data flow extractor executes the miniature program in local, and then replays it on cloud. Through this, we get the performance difference between local and cloud, in terms of CPU, memory, hard disk, and communications [7].

In this framework, we adopt the performance driver architecture, which transforms a software structure model to a performance model. The performance model contains two components: a platform-independent model and a platform-dependent model. The platform-independent model focuses on logic view, which is independent of the concrete implementation and platform support. While the platform-dependent model shows the underlying technology and the related platform-based implementation details. We use the following performance key model as the final performance prediction model:

$$\Gamma = \alpha\beta M + C(\alpha > 0, \beta > 1) \tag{1}$$

where Γ denotes the performance prediction value of the program, M is the time cost of the miniature program executed in local environment, C is the communication cost, α is the difference degree between local and cloud, β is the scaling coefficient between the real program and the miniature program. In the following, we will present the measurement method for M (in Section IV and V) and C (in Section V).

IV. PERFORMANCE MODEL EXTRACTION

The control flow is the output of the decomposition engine component, and it is used to get the platform-dependent model and the platform-independent model. Below we describe each part in details.

A. Lexical Analysis and Syntax Analysis

Lexical Analysis: The purpose of lexical analysis is to identify words in a sentence and mark them with syntactic tagging. We use the GNU GCC compiler as our lexical analyzer. In the prototype system, we capture the interim result strings of the *_cpp_lex_direct* function and we use string matching to identify performance-related words (e.g., *while*, *for*, *switch*), which are stored as records in a small database [11, 12].

Syntax Analysis: The syntax analysis is to check words given by the lexical analysis and verify whether a given sequence of symbols is a correct sentence. In this phase, we rewrite the *c_parser_translation_unit* function, which grasps the input of every performance-related word, and appends it to the corresponding record in the database .

Algorithm 1 Extracting scaling basic block algorithm

```
Input: SC, the source code of application
Output: < $sbb_i$, $d_i$ >
        sbb, scaling basic block; $d_i$, the input of $sbb_i$
1:while( SC )
2:     do lexical analysis
3:     getting loop block
4:     sbb ← loop block
5:     SBBs ← SBBs ∪ sbb
6:end while
7:while( SBBs )
8:     do syntax analysis
9:     getting $d_i$ of $sbb_i$
10:    <$sbb_i$, $d_i$ >
11: end while
```

Algorithm 1 shows the pseudo-code of extracting scaling basic blocks. The algorithm includes two main steps. Line 1-6 find loop blocks by scanning the application source code (i.e., lexical analysis). Based on the communication pattern, line 7-11 extract the dataset scale about each basic block, and denotes it as <sbb_i, d_i>. Consequently, by lexical analysis and syntax analysis, performance-dependent but semantics-independent model can be built. Outputs of lexical analysis and syntax analysis only describe each separate performance block of the application, but don't have the context information of the blocks.

B. *Dependence Analysis*

In compiler theory, dependence analysis produces execution-order constraints between statements and instructions. Broadly speaking, a statement S2 depends on S1 if S1 must be executed before S2. In general, there are two classes of dependencies: control dependence and data dependence [11].

Dependence analysis determines whether or not it is safe to reorder or parallelize statements. There are two main application dependences.

Control dependence: Control dependence is a situation in which a program's instruction executes if the previous instruction evaluates in a way that allows its execution. Loop dependence is the most important, because the size of a loop structure determines the main time cost in a parallel computation program. Loop dependence analysis is to determine whether statements within a loop body form dependence, with respect to array access, modification, induction, reduction, private variables, simplification of loop-independent code and management of conditional branches inside the loop body.

Data dependence: Data dependence is a situation where the attributes of data (e.g., input/output size, type, memory location, data function) affect the program. Through data dependence analysis, we can get the time cost of each aspect (e.g., CPU, memory, hard disk, communication) for executing the program, and the performance different between local and cloud.

The goal of dependency analysis is to find the system core structures, frequent patterns. Control dependence analyzes solutions based on static analysis and uses source-code-based data in the form of execution traces. An execution trace can be a function, procedure, or method being called. Execution traces are collected using techniques such as source code instrumentation, platform profiling, and compiler profiling. Most techniques and tools for execution trace analysis are designed for specific paradigms and even specific programming languages. While data dependence analysis solutions are based on dynamic analysis and execute the miniature version of the program, which is obtained by control dependence analysis. Through control dependence analysis, we can get the time cost which is proportional to that of the real program, and use the replayer to get the run-time environment differences between local and cloud.

V. Performance Equal Scaling Metric

Computation-intensive applications place unique and distinct demands on computing resources. In this section, we present an efficient method to predict the processing time of an application on cloud by performance equal scaling instead of actual migration.

A. *Computation Proportional Scaling*

Large computation-intensive applications running on cloud are often hard to monitor for a variety of reasons. Several technical challenges still exist. First, the platforms themselves are complex systems of heterogeneous nodes, and the platforms differ widely among themselves. Second, because of the heterogeneity, virtual machines (VMs) make resource access on cloud by virtualization of heterogeneous resources.

Due to virtualization, important factors of running time, such as memory size, memory frequency, and hard disk speed, can't be measured directly. We divide the factors into three classes: CPU-related factors, memory-related factors and IO-related factors. These factors affect computation time in different ways. The combined effect of these factors on computation time is hard to determinate. In cloud, VMs and host OSs shield the factors from customers. Hence, we have to measure cloud resources using trace ideas by running the application in real environment. To get the performance of an application, the first method that comes to mind is to replay it on cloud with small dataset and count the system calls. However, this method is not feasible because there are too many system calls to collect, which would severely affect system performance.

As stated in section IV, the scaling block, to a certain extent determines the computation of an application. However, scaling block only describes the logical structure of an application, which is not related to the actual running environment and dataset scale. In this case, we need to build a performance model that is related to the cloud environment, and we call it the context-sensitive performance model. We run an application with small dataset scale in local and then replay it on cloud, and then we can get the performance scale ratio between local and cloud.

The context-sensitive performance model is based on a workload-independent ratio, which extracts parameters from the comparison between running in the local and cloud with a small dataset scale. The purpose of replaying in cloud is to get the impacts from practical parameters of the cloud.

B. *Communication Cost*

Most computation-intensive applications use the Message Passing Interface (MPI) framework for achieving their parallel computation on different VMs [2, 10]. The communication overhead is only related to the amount of socket communications, and it doesn't depend on the dataset scale of the application.

Compared to the computation cost, the communication cost is much easier to predict. The communication cost typically includes latency and bandwidth, which are determined by the network being used. There are many existing programs to monitor MPI message communications. MPI communication and network latency are not related to the dataset scale. Because all MPI communications use socket, we only need to monitor socket communications on cloud platform. We also know the run-time environment difference between local and cloud; hence we can predict the communication cost on cloud.

C. *Performance Prediction*

We need to predicate both the computation and communication cost.

Computation: By lexical analysis and syntax analysis, we can get the performance miniature version of an application. However, the dataset scale of application has different effects on each block. Hence, we need to obtain the relationship between dataset scale and the input of each basic scaling performance block. The solution is based on the coverage test technique. It makes static analysis on the source code, which is inserted with stub code at the beginning and end of each function.

Communication: As mentioned earlier, we only need to monitor the MPI socket communications of all VMs when replay the application in cloud with small scale dataset. MPI socket has almost the same cost while running with large scale dataset [2]. Hence, it doesn't need monitor MPI socket on local platform, because local and cloud network environments are different. Our goal is to predict the application performance in cloud, so we need to run the application using the cloud network rather than the local network. Therefore, MPI communication monitoring must be done by running the application in cloud with small scale dataset.

VI. Evaluation

To evaluate the performance of our strategy, we run real computation-intensive applications in both local platform and cloud platform, and we compare the results between the two platforms.

A. HPL Benchmark Case

As a computation-intensive application, HPL is a Linpack benchmark package widely used in massive cluster system performance test [2]. The HPL algorithm is designed to solve a linear system by LU factorization with row partial pivoting. *N* is the order of coefficient matrix *A*. We use an open source cloud Eucalyptus [15] the cloud platform, which contains 32 virtual machines sharing 4 physical machines, and the bandwidth of inter-network in cloud is 100MB. The local platform is a PC with Intel T5500 CPU, 2G RAM and 7200 RPM hard-disk. In the following subsections, we run a serial of experiments with different dataset scales.

B. Detailed Analysis

1) Platform-related Factor Difference

First we compare the performance difference between the local and cloud platforms. There are three key factors that are critical to computation performance on a specific platform: CPU-related factor, memory-related factor and I/O-related factor. We use a suite of benchmark applications that test various aspects of the computing infrastructure offered by cloud. There are traditional computation performance benchmark suites for measuring the three factor, such as the busy-loop, memory-intensive benchmarks, and I/O benchmarks. We inject timing function to HPL and replay it in cloud [12], and then we can obtain the performance ratios between local and cloud by dynamic analysis.

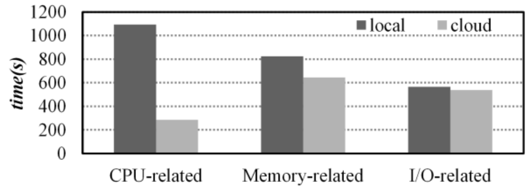

Figure 2. Performance difference between local and cloud

From Figure 2, we can see that the CPU and memory related performance ratios between the two platforms are quite different. While, I/O related performances on the two platforms are almost the same. Parameter α is the difference degree between local and cloud, and it is determined as follows:

$$\alpha = CPU_r CPU_p + Mem_r Mem_p + IO_r IO_p \qquad (2)$$

where CPU_r, Mem_r and IO_r denote the ratio of three performance-dependent factors between local and cloud respectively. While CPU_p, Mem_p and IO_p denote the proportion of three factors in the application, respectively.

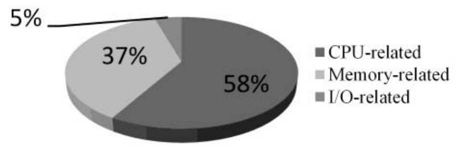

Figure 3. Performance factor proportion of HPL

To evaluate the predication performance of our method, we deploy the HPL benchmark with different dataset scales on local and then replay it in cloud. Figure 3 shows the percentage of every factor in HPL. The result in Figure 3 is obtained by using a few small scale datasets, i.e., the $N \times N$ HPL benchmark, where $N_{small} = \{4000, 4100, \dots, 6000\}$.

For different orders of the coefficient matrix *A*, HPL requires different running time to execute the matrix LU decomposition. The floating-point execution time varies when problem size changes. Computation-intensive applications tend to consume a lot of memory. When there is not enough memory, other programs that reside in memory have to be swapped out to the hard disk. To reduce the effect of memory swapping, we set the page swap rate under small datasets the same as that under large datasets.

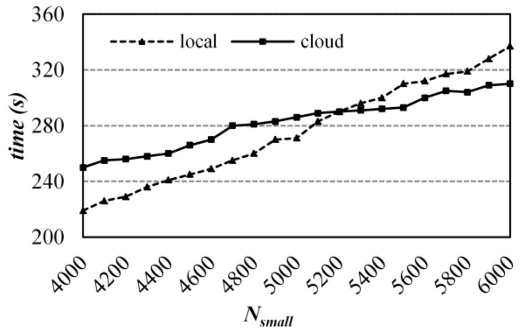

Figure 4. Computation cost of HPL on local and replaying in cloud

Figure 4 plots the running time of HPL on local and cloud platforms using N_{small}. The results show that: when the dataset is small, the application running time in cloud is larger than that on local. This is consistent with theoretical analysis: when the dataset is small, the communication cost is the majority of the overall cost. Different from local platform, the cloud platform runs an application in several VMs, and hence the communication cost becomes the bottleneck. When dataset increases, the computation cost becomes the major cost. Because cloud has more computing power, the application running time in cloud is smaller than that on local.

Parameter β is the scaling coefficient between a real program and its miniature program, and β is given by

$$\beta = \frac{n_{real}}{n_i} \quad (n_i \in N_{small}) \qquad (3)$$

In our test, the communication cost C only includes the MPI cost. The computation cost has a nonlinear relationship with the dataset size. The lexical analysis component can extract the performance skeleton. For different applications, the dataset size has different effects on scaling blocks.

2) Performance Prediction and Accuracy

In this subsection, we evaluate how the prediction accuracy changes under datasets with different scales. For this purpose, we predict the running time and communication time on the cloud for each dataset, and then we also use experiments to measure the time. We use N_{small} datasets to predict the performance of $n_{real} = 15000$ and $n_{real} = 30000$ and check the accuracy of equation (1).

Figure 5 plots the real and predicted running time of HPL on N_{small}. The baseline is the actual performance overhead of HPL with n_{real} dataset on cloud. Figure 5 shows that our scheme can accurately predict the HPL performance, and the error is in the range of 5.09%~1.43% when $n_{real} = 15000$ as shown in Figure 5(a), while 4.07%~0.98% when $n_{real} = 30000$ as shown in Figure 5(b). There are two reasons for this good result. First the cloud replayer uses the same performance model and issues the same network MPI calls as the application on local. Second, our scheme extracts the ratios of CPU usage, memory usage and I/O usage before replaying on cloud. We also find that the prediction inaccuracy decreases when the dataset scale gets closer to n_{real}. It is interesting to note that the prediction accuracy depends on the datasets used for prediction. The closer the small dataset scale to the large one, the higher the accuracy.

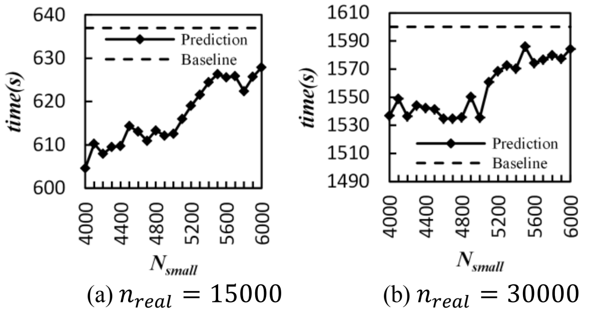

Figure 5. Performance prediction inaccuracy by N_{small}

To sum up, the experimental results show that our scheme can accurately predict the performance of computation-intensive applications on cloud with small dataset scale.

VII. Related Work

Li *et al.* [7] propose using architecture independent characteristics to find the most similar benchmarks, which are used to predict the performance of CPU-intensive applications across a large collection of CPU types. On the other hand, our work captures the active running time using lightweight technique, and we use performance scaling block to predict the running time on cloud.

In [8], Li *et al.* compares performance of multiple CSPs. This work focuses on how much cloud computation a client should buy, and it tries to predict the performance of computation-intensive applications. Our work uses the performance equal scaling strategy instead of simulations to predict the running time on cloud. Four popular commercial cloud providers are compared in [13], which finds that the performance and costs of various CSPs differ significantly.

VIII. Conclusion

In this paper, we presented a performance prediction scheme for computation-intensive applications on cloud. We identified and addressed two key challenges: (1) how to find the application performance scaling blocks, and (2) how to predict computation-intensive application performance using small dataset scale. Our real-network experiments showed that the scheme can achieve accurate predictions with low overhead. Our performance prediction scheme could help a cloud customer estimate the cost of running a computation-intensive application on cloud without actually deploying it.

Acknowledgment

This work is supported by the project of National Natural Science Foundation of China (60903166, 61100188, 61173144), National Basic Research Program (973 Program) of China (2011CB302605) and National High Technology Research and Development Program (863 Program) of China (2010AA012504), and by the US National Science Foundation under grants CNS-0963578, CNS-1022552 and CNS-1065444.

References

[1] *Amazon Relational Database Service (Amazon RDS)* [Online]. Available: http://aws.amazon.com/rds/

[2] *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers* [Online]. Avaliable: http://netlib. org/benchmark/hpl/

[3] G. H. Wang, T. S. E. Ng, "The Impact of Virtualization on Network Perfor-mance of Amazon EC2 Data Center," in *Proc. 29th IEEE Int. Conf. Comput. Commun.*, San Diego, CA, 2010, 1-9.

[4] *CloudCmp Project Website* [Online]. Available: http://cloudcmp.net

[5] *Google AppEngine* [Online]. Available: http://code.google.com/ appengine

[6] M. Hajjat, X. Sun, *et al.,* "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud," in *Proc. 2010 ACM Int. Conf. Special Interest Group on Data Commun.*, New Delhi, India, 2010, 243-254.

[7] A. Li, X. Zong, *et al.,* "CloudProphet: Towards Application Performance Prediction in Cloud," in *Proc. 2011 ACM Int. Conf. Special Interest Group on Data Commun.* Toronto, ON, Canada, 2011, 426-427.

[8] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Shopping for a Cloud Made Easy," *in Proc. 2nd UNSEIX Workshop on Hot Topics in Cloud Computing*, Boston, MA, USA, 2010.

[9] E. Walker. "Benchmarking Amazon EC2 for High-Performance Scientific Computing," *USENIX Login*, vol.33, no 5, Oct. 2008.

[10] R. A. Balance, J. Cook, "Monitoring MPI Programs for Performance Characterization and Management Control," *In Proc. 28th ACM Symp. on Applied Computing*, 2010, 2305–2310.

[11] *GCC, the GNU compiler Collection* [Online]. Available: http://gcc.gnu.org

[12] *GLIBC, the GNU C Library* [Online]. Available: http://www.gnu.org/ software/libc

[13] A. Li, X. W. Yang, S. Kandula, M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proc. 2010 Internet Measurement Conf.*, Melbourne, Australia, 2010, 1-14.

[14] M. P. Mesnier, M. Wachs, R. R. Sambasivan, *et al.,* "//trace: parallel trace replay with approximate causal events," *in Proc. 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, 2007.

[15] *Eucalyptus* [Online]. Available: http://open.eucalyptus.com