

# Conductor: Orchestrating the Clouds

Alexander Wieder<sup>†</sup>   Pramod Bhatotia<sup>†</sup>   Ansley Post<sup>†‡</sup>   Rodrigo Rodrigues<sup>†</sup>  
<sup>†</sup>Max Planck Institute for Software Systems (MPI-SWS) and <sup>‡</sup>Rice University

## ABSTRACT

Cloud computing enables customers to access virtually unlimited resources on demand and without any fixed upfront cost. However, the commoditization of computing resources imposes new challenges in how to manage them: customers of cloud services are no longer restricted to the resources they own, but instead choose from a variety of different services offered by different providers, and the impact of these choices on price and overall performance is not always clear. Furthermore, having to take into account new cloud products and services, the cost of recovering from faults, or price fluctuations due to spot markets makes the picture even more unclear.

This position paper highlights a series of challenges that must be overcome in order to allow customers to better leverage cloud resources. We also make the case for a system called Conductor that automatically manages resources in cloud computing to meet user-specifiable optimization goals, such as minimizing monetary cost or completion time. Finally, we discuss some of the challenges we will face in building such a system.

## Categories and Subject Descriptors

D.4.7 [Organization and Design]: Distributed systems

## General Terms

Management, Economics, Performance

## Keywords

cloud computing, resource management, MapReduce, optimization, spot markets

## 1. INTRODUCTION

Cloud computing gives programmers access to almost unlimited resources at any given moment in time. This allows users and organizations to dynamically adapt the resources

used to their problem, without requiring them to invest in permanent IT infrastructure. This ease of scalability has made cloud computing popular among end users and a subject of excitement in both research and industry. Users now have the opportunity to move computations into the cloud that would have otherwise be impossible or too expensive to perform locally.

These new opportunities, however, raise new challenges. In the past, organizations invested in building and maintaining a certain IT infrastructure, and given that investment they could estimate how long a certain computation would take (or even if it was feasible). In the new cloud computing world, however, it is possible to spend an almost unbounded amount of money acquiring seemingly infinite resources. This changes the nature of the equation, since now organizations must estimate the cost of a computation, and choose how many resources should be invested in it. Ideally, a user would invest the exact amount that is needed to satisfy their computation goals, but no more.

The picture is further muddled by the fact that cloud computing services provide many different resources. For example, EC2 provides eight different types of virtual machine instances, and it is unclear how a computation's performance will change if run on different instance types. In addition to the rental of a virtual machine, cloud providers also charge for storage and for bandwidth consumed between the cloud and the outside world. Given these factors, without significant analysis before deployment it is hard to know ahead of time the exact cost of a cloud deployment. Furthermore, this analysis is complicated by factors such as dynamic pricing due to spot markets, or the need to account for the possibility of faults, and factor in that different resources might incur different costs for fault recovery.

This position paper advocates a system that will automatically select and acquire cloud computing resources. We begin by highlighting a number of reasons why this is a challenging problem and illustrate this fact by a real-world example. We then propose Conductor, a sketch of a system for automating resource management in cloud computing for customers of cloud services. With this system, we simply ask the user to specify simple goals, such as a budget or a deadline for the computation, as well as some simple information about the computation. We then try to determine an execution plan that selects the appropriate resources to ensure that the computation completes before the deadline or below the budget, if possible. Our approach will enable us to do this even in the presence of a large variety of possible resources, as well as changes in the availability of resources and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

LADIS '10 Zürich, Switzerland

Copyright 2010 ACM 978-1-4503-0406-1 ...\$10.00.

the pricing of those resources. This system, once deployed, will help programmers use cloud computing infrastructure more easily, and for less money.

## 2. MOTIVATING EXAMPLE

In order to motivate both the problem and the solution presented in this paper, we first present an example based on a real-world utilization of cloud services, illustrating that for a very simple task there are several alternative deployment strategies with different costs. In 2007, the New York Times converted 11 million scanned articles published between 1851 and 1980 to the PDF format [3]. The scanned articles added up to 4TB of data, which was uploaded to S3, Amazon’s key-value storage service. The conversion was performed using a MapReduce job that was executed on 100 EC2 nodes and took less than 24 hours to complete. The generated PDF files were approximately 1.5TB in total size and were stored in S3.

We begin by producing a rough estimate of the monetary cost of this conversion based on today’s prices for Amazon’s cloud services. Given that the computation using EC2 nodes took approximately 2400 machine-hours, the actual processing of the articles yielded a cost of \$204, assuming the use of small EC2 instances. To this cost we must add the cost for storing the full input and output data on S3, which amounts to \$216, and a cost of \$230 for network traffic for downloading the result. (Data upload was free at the time of writing.) These costs assumed that the customer was connected to the cloud by a 100 Mbit/s uplink, for the purpose of determining how long data was stored in S3 for. Therefore, the cost for running this job using S3 and EC2 adds up to \$650.

It is interesting to note that usage of S3 accounts for a large fraction of the overall cost. This leads us to study alternatives for storing data, namely storing it locally on EC2 instances. Each small EC2 instance is equipped with a virtual disk of 160GB, so the total capacity of the 100 nodes rented would be sufficient for storing all the input and output data. By using the local storage of EC2 instances instead of S3 we were able to eliminate the second highest cost item. However, this approach also requires EC2 instances to be running for the entire duration of data upload and download, thus incurring an additional cost that varies according to the rate at which data is transferred to or from these instances.

To quantify the cost of this alternative execution strategy, we analyzed its impact on resource utilization. In our analysis we also took into account that parts of the input data can be processed while other parts are still being uploaded. We found that the cost for S3 could be eliminated by using the local disks of EC2 nodes. Assuming once more an upload speed of 100 MBit/s, our analysis revealed that we require an additional 230 EC2 machine-hours (at an extra cost of less than \$20) for storing data while it is being uploaded and downloaded. The total cost for running the job using this strategy is \$455 (\$225 for EC2 nodes, \$230 for network traffic). Compared to storing data on S3, we were able to cut costs by almost one third.

This alternative strategy for storage is not always advantageous. In fact, small variations in the deployment characteristics can affect which option is best. For instance, consider a scenario in which the customer in the above example had a connection speed of 20Mbit/s instead of 100Mbit/s, but the other parameters were identical. In this case, up-

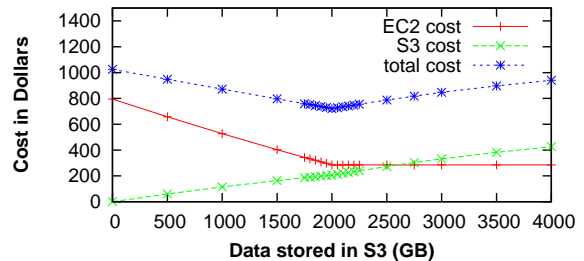


Figure 1: Total cost of example computation while varying the fraction of input data stored on S3 versus EC2 nodes (assuming a 20Mbps upload link).

loading and downloading data to and from EC2 instances requires instances to run for an additional 6100 machine-hours. During this time instances would not perform computations because the transfer rate is slower than the rate at which data is consumed by the MapReduce job. As a consequence, this scenario requires approximately 8500 EC2 machine-hours, of which only 2400 machine-hours are spent processing the input. Thus the overall cost for executing the job with the alternative execution strategy is \$1025 (\$795 for EC2 nodes, \$230 for network traffic).

Clearly the overall cost of this job depends on the amount of data stored in S3 versus on the local disks of EC2 nodes. The influence of the amount of data stored on S3 on overall cost is depicted in Figure 1. Notably, as shown in the figure, the minimum cost for this job is achieved neither by storing all input data on S3 nor locally on EC2 nodes. Rather, the (cost) optimal strategy is storing approximately 2TB of input data on S3 and using the local disks of the EC2 instances for the remaining data. This result illustrates that the optimal way of utilizing the available resources depends on the characteristics of the job, the pricing model, and the network speed. There is no obvious one-size-fits-all solution for utilizing these resources optimally.

## 3. RESOURCE MANAGEMENT CHALLENGES

The preceding example showed that, even for a simple example, it is difficult to choose from the different resources that a cloud provider makes available. In this section, we elaborate on several factors that contribute to the difficulty of making decisions about resource selection when outsourcing computations to the cloud.

**Resource and provider diversity.** Cloud customers must choose among a variety of resources with different price and performance characteristics. This makes it hard to determine the optimal resource utilization strategy for a particular task. Furthermore, the possibility of using spare resources, such as the local disk of EC2 nodes that run a computation, makes it even harder to decide which resource to use.

**Dynamic Pricing.** The pricing for cloud services can vary over time as providers adjust their pricing models, or when new providers emerge. In addition, recently, spot markets for cloud computing services have been introduced (e.g., dynamic pricing for Amazon’s EC2 instances). Spot markets bring new opportunities, as well as new challenges for customers. Customers may choose to delay tasks until the spot

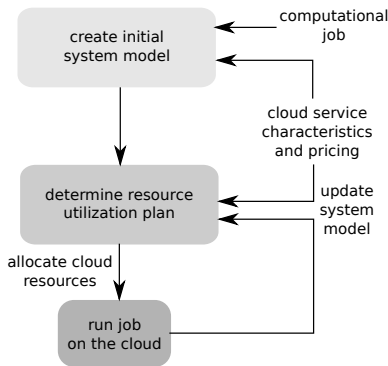


Figure 2: System overview

market price is low. On the other hand, resource utilization must be adjusted to price at runtime, thus making management more difficult.

**Faults.** The reliability of storage and computation is of particular importance when using cloud computing services for executing jobs based on higher-level languages like Pig [9]. Pig programs compile down to multi-staged MapReduce computations in which the result of one stage is used as the input of the subsequent stage. When intermediate results become unavailable due to data loss, they must be recomputed by re-executing all previous stages. The cost for recovery hence depends on the number and complexity of the previous stages, and generally increases as the computation progresses [6]. Therefore, when the reliability characteristics of the available storage options differ, it is important to factor in the cost of recovery when computing the total cost of each option.

**Tightly coupled data and computation.** Although cloud computing providers offer separate services for storage and computation, computation is often tightly coupled with its input and output data: to perform a computation, the input data must be transferred to where the computation is performed. A consequence for resource management in cloud computing is that the time and cost this transfer incurs must be considered in determining a plan for resource utilization. The coupling between data and computation also prevents simplistic resource management approaches, such as always using the cheapest one – this could lead to increasing the overall cost or the completion time because of the cost of transferring the data between computation and storage locations.

All of the above issues raise the bar for determining the right set of cloud resources to choose from. Next, we outline how we intend to tackle this problem in a way that addresses these issues.

## 4. CONDUCTOR: CHOOSING CLOUD COMPONENTS

In this section we give an overview of our solution and describe some of the challenges of building such a system.

We propose a system called Conductor that takes the following input: (1) a computation to be executed in the cloud, (2) a set of cloud services that could be used for executing the computation, and (3) a set of goals to optimize the execution for, such as minimizing execution time for a given monetary budget, minimizing cost for a given time budget,

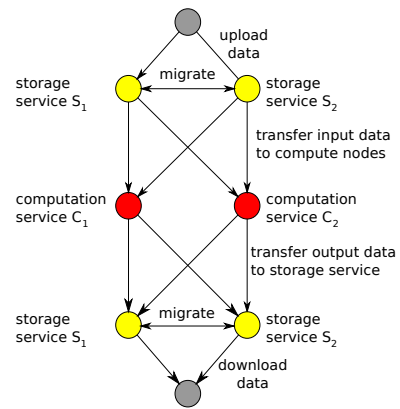


Figure 3: System model example

or some combination of the two.

Given these inputs, the goal of Conductor is to find an execution plan that selects the resources to best meet the goals specified by the customer. Furthermore, the system should be able to deploy and execute this plan, as well as adapt to changing conditions, such as dynamic pricing or node failure.

At a high level, the way we achieve this goal can be described as follows (as depicted in Figure 2).

1. Model the computation, the set of resources available from various cloud computing providers, their cost, and their performance.
2. The node responsible for planning, typically under control of the customer, determines the optimal job execution plan automatically, by using a solver.
3. Deploy the planned execution and monitor the execution to identify conditions that constitute possible deviations from the original model.
4. Feed the new conditions back to the model, and compute a new plan, and alter the deployment accordingly.

In the following sections we describe challenges arising in each of these steps, and sketch an approach to address them.

### 4.1 Modeling computations

Our approach to modeling computations is to partition the computational job and the cloud services by the types of resources that are consumed or offered (e.g., computing versus storage resources), and use a graph to model the dependencies between stages of the computation as shown in Figure 3.

Each step of the computation job is represented as a vertex, and edges between these computation vertices and storage vertices represent the possible data flow of the job for a particular resource. Edges are annotated with monetary cost and time needed for the computation nodes to access the storage service, and these costs can be derived from service descriptions that are offered by cloud providers. Data migration between two storage services is represented as an edge between corresponding vertices, and is annotated with the monetary cost and time for migration.

The output data size as well as processing performance of the computation nodes can initially be estimated, and

during runtime the model can be updated with extrapolated measurements obtained during the execution. These measurements can be performed by each computation node during runtime. Nodes can monitor the processing speed as well as the output data size in each phase of the job. Similarly, the network transfer performance between computation nodes and storage vertices can be sampled. The results of these measurements can then be transmitted to the node that performs the planning, processed using statistical methods, which enables the model to be updated accordingly.

A primary challenge is how to predict data access patterns of computations, which is needed to model data flow as described above. To start, we consider only applications based on MapReduce, which, given the popularity of this computing paradigm, covers a large class of useful applications. This enables us to leverage the fact that these jobs exhibit a data-parallel structure for which the data-flow can be pre-computed. The details about how we model MapReduce computations using linear programming appear in a separate document [13].

## 4.2 Programming abstractions

Once Conductor finds an optimal execution plan for a model of a job and available resources, it deploys the plan by instantiating the appropriate cloud services. The challenge is that different services may have different storage and computation interfaces. To address this, we will provide a uniform interface via abstraction layers for storage and computing resources. This allows us to transparently manage the resources according to the execution plan and hide complexity from the application.

For storage, we envision a key-value storage service, similar to Amazon’s S3. Key-value-storage is generic enough for a range of other abstractions to be built on top of it. For instance, there are already multiple file system implementations built on key-value stores [8]. Also, many existing applications already support the use of S3 as data storage, so they will need only minor modifications when using our system. This is the case, in particular, of the Hadoop open source implementation of the MapReduce programming paradigm. Since storage is abstracted we can easily replace key-value storage in S3 with local storage from the VM instances.

For computation it is more difficult to create an abstraction layer for different services. For instance, it is easy to provide a MapReduce computation service (like Amazon’s Elastic MapReduce service) on top of a virtual machine abstraction, but the inverse is not true. Since our initial approach is to restrict ourselves to MapReduce computations, we can use a MapReduce service as our abstraction for computational resources, and still incorporate different service providers that offer VM-based computation services.

## 5. RELATED WORK

Automatic resource management has been studied in other contexts. For example, operating systems automatically allocate resources; cluster resource management systems have been proposed [2, 4]; and resource management in Grid computing has also been studied [7, 11]. In cloud computing, resource management presents new challenges that are not addressed by previous work. Namely, as opposed to clusters and grids, a cloud computation has access to infinite resources but must pay an increased marginal cost for each

resource it uses. In addition, cloud computing exposes heterogeneous resources to the client. Our system takes these distinguishing features into account, and modeling them is one of the primary challenges of this work.

Rhizoma [14] proposed automating resource allocation for generic applications. Rhizoma also uses a specification of resources and maximizes the utility for an application. Although Rhizoma uses cloud computing as motivation, the application they describe is that of a publish-subscribe system deployed on PlanetLab, where the challenge is to find well-connected, lightly loaded nodes. Unlike Rhizoma, our proposal is geared towards MapReduce computations, without requiring the specification of application resource requirements.

There exist tools that are able to simulate the execution of MapReduce computations in a given cluster setup [12]. These take into account properties like the network topology, hard disk characteristics, and system configuration parameters. However, in cloud computing, many of these factors are not visible to customers, and beyond their control. Also, these tools are oblivious to the economics and dynamics of cloud computing resources. Still, we envision leveraging some of these modeling and simulation techniques for our system in order to optimize configuration parameters of the MapReduce implementation.

We model the problem of cloud resource allocation as a linear program. Modeling other problems in such a way has been done a variety of fields including systems research [5]. A reason for using optimization is that it allows us to adapt to dynamic markets for computation (such as the spot instances of Amazon EC2) and to characterize the properties of diverse resources. Similarly, a recent proposal [10] seeks to shift computations among multiple data centers based on changing electricity prices in the spot market.

Dynamic allocation of spot instances for MapReduce computations has also been proposed recently [1]. In contrast to this approach, we focus on the broader problem of trying to incorporate multiple providers of potentially diverse resources (both from regular and spot markets) to determine a globally optimal resource allocation plan.

## 6. DISCUSSION

In this section we discuss some obstacles and opportunities that are left open by the paper.

**Economic incentives:** Our system allows customers to adapt resource usage to the prices available at a given time. If many users adopt our system it may lead to a change in cloud computing economics. Our system could make the market for computation more efficient, as price changes will stimulate demand when clients adapt to lower costs. This may lead undesirable side effects for the provider, such as lower revenue. A consequence of this may be that providers seek to lock in users by offering low introductory rates for storage and later increase data transfer costs to prevent migration of data to other providers. Providers may also lower prices for users willing to commit to a fixed price contract. We see this leading to a cloud computing market that more resembles the market for commodities such as electricity [10], where unused resources are offered at lower prices, and heavy users adapt to the excess supply by increasing demand.

**Generalization of computing model:** In this paper our discussion focused on a restricted model of computa-

tion, that of MapReduce. Going forward we seek to generalize our techniques to other types of cloud computing usage scenarios. For example, we aim to support applications ranging from computation intensive simulation applications, to network services such as web applications. The characteristics of these applications in terms of workload, performance goals, computation predictability all very different. Successfully applying our techniques to a more general class of applications may require developing new models to predict application behavior, or require application developers to add instrumentation to support better monitoring.

## 7. CONCLUSION

In this paper we motivated and sketched the design of Conductor, a system that assists cloud customers in choosing the right set of resources to use when outsourcing computations to the cloud. The premise underlying this paper is that the selection and management of cloud resources is a complicated task, that will only get worse as cloud providers join and leave the market, and the offer of cloud services becomes ever more diverse. Anecdotally, after the submission of this paper, Amazon announced a reduced redundancy storage service that promises to offer less expensive storage but with lower availability and durability targets. This exemplifies the problem that this paper is considering: enabling customers of Amazon's storage services to more easily determine if it is worth migrating its computations to take advantage of this new option, and also providing mechanisms to do so seamlessly.

To address this set of problems we propose Conductor, a system that automates the task of choosing and managing the right set of cloud resources for a class of cloud applications. Conductor only requires users to specify simple goals and some information about the computation, and uses optimization tools to determine an execution plan, which can be adapted to the actual deployment conditions. Conductor is an important first step in the direction of improving the selection of cloud resources and hiding heterogeneity and management complexity from users. Yet, the current design still has important limitations, notably being restricted to MapReduce computations, which have a predictable resource access pattern. Generalizing the set of supported computations and making the system even more adaptive to their behavior is an important avenue of future work.

## 8. REFERENCES

- [1] CHOCHAN, N., CASTILLO, C., SPREITZER, M., STEINDER, M., TANTAWI, A., AND KRINTZ, C. See Spot Run: Using spot instances for MapReduce workflows. In *2nd USENIX Workshop on Hot Topics in Cloud Computing* (June 2010).
- [2] DUSSEAU, A. C., ARPACI, R. H., AND CULLER, D. E. Effective distributed scheduling of parallel workloads. In *SIGMETRICS '96: Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (1996), pp. 25–36.
- [3] GOTTFRID, D. Self-service, Prorated Super Computing Fun!  
<http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/>.
- [4] ISARD, M., PRABHAKARAN, V., CURREY, J., WIEDER, U., TALWAR, K., AND GOLDBERG, A. Quincy: fair scheduling for distributed computing clusters. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), pp. 261–276.
- [5] KEETON, K., KELLY, T., MERCHANT, A., SANTOS, C., WIENER, J., ZHU, X., AND BEYER, D. Don't settle for less than the best: Use optimization to make decisions. In *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems* (May 2007), pp. 1–6.
- [6] KO, S. Y., HOQUE, I., CHO, B., AND GUPTA, I. On availability of intermediate data in cloud computations. In *Proceedings of HotOS'09: 12th Workshop on Hot Topics in Operating Systems*.
- [7] KRAWCZYK, S., AND BUBENDORFER, K. Grid resource allocation: allocation mechanisms and utilisation patterns. In *AusGrid '08: Proceedings of the sixth Australasian workshop on Grid computing and e-research* (Darlinghurst, Australia, Australia, 2008), Australian Computer Society, Inc., pp. 73–81.
- [8] MUTHITACHAROEN, A., MORRIS, R., GIL, T. M., AND CHEN, B. Ivy: a read/write peer-to-peer file system. *SIGOPS Operating Systems Review* 36, SI (2002), 31–44.
- [9] OLSTON, C., REED, B., SRIVASTAVA, U., KUMAR, R., AND TOMKINS, A. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), pp. 1099–1110.
- [10] QURESHI, A., WEBER, R., BALAKRISHNAN, H., GUTTAG, J., AND MAGGS, B. Cutting the electric bill for internet-scale systems. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (2009), pp. 123–134.
- [11] RAMAN, R., LIVNY, M., AND SOLOMON, M. Matchmaking: Distributed resource management for high throughput computing. In *HPDC '98: Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing* (Washington, DC, USA, 1998), IEEE Computer Society, p. 140.
- [12] WANG, G., BUTT, A. R., PANDEY, P., AND GUPTA, K. A simulation approach to evaluating design decisions in mapreduce setups. In *MASCOTS '09: Proceedings of the International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*.
- [13] WIEDER, A., BHATOTIA, P., POST, A., AND RODRIGUES, R. Brief Announcement: Modelling MapReduce for Optimal Execution in the Cloud. In *PODC '10: Proceedings of the 29th ACM symposium on Principles of distributed computing*.
- [14] YIN, Q., SCHÜPBACH, A., CAPPOS, J., BAUMANN, A., AND ROSCOE, T. Rhizoma: a runtime for self-deploying, self-managing overlays. In *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware* (2009), pp. 1–20.