Optimizing Utility in Cloud Computing through Autonomic Workload Execution

Norman W. Paton, Marcelo A. T. de Aragão, Kevin Lee, Alvaro A. A. Fernandes, Rizos Sakellariou
School of Computer Science, University of Manchester, U.K.
(norm,maragao,klee,rizos,alvaro)@cs.man.ac.uk

Abstract

Cloud computing provides services to potentially numerous remote users with diverse requirements. Although predictable performance can be obtained through the provision of carefully delimited services, it is straightforward to identify applications in which a cloud might usefully host services that support the composition of more primitive analysis services or the evaluation of complex data analysis requests. In such settings, a service provider must manage complex and unpredictable workloads. This paper describes how utility functions can be used to make explicit the desirability of different workload evaluation strategies, and how optimization can be used to select between such alternatives. The approach is illustrated for workloads consisting of workflows or queries.

1 Introduction

Cloud computing essentially provides services; shared computational resources execute potentially diverse requests on behalf of users who may have widely differing expectations. In such a setting, someplace in the architecture, decisions have to be made as to which requests from which users are to be executed on which computational resources, and when. From the perspective of the service provider, such decision making may be eased through the provision of restrictive interfaces to cloud services, as discussed for cloud data services in the Claremont Report on Database Research [1]:

> Early cloud data services offer an API that is much more restricted than that of traditional database systems, with a minimalist query language and limited consistency guarantees. This pushes more programming burden on developers, but allows cloud providers to build more predictable services, and to offer service level agreements that would be hard to provide for a full-function SQL data service. More work and experience will be needed on several fronts to explore the continuum between today's early cloud data services and more full-functioned but probably less predictable alternatives.

This paper explores part of this space, by describing an approach to workload execution that is applicable to different types of workload and that takes account of: (i) the properties of the workload; (ii) the nature of the service level agreement associated with user tasks; and (iii) competition for the use of finite, shared resources.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

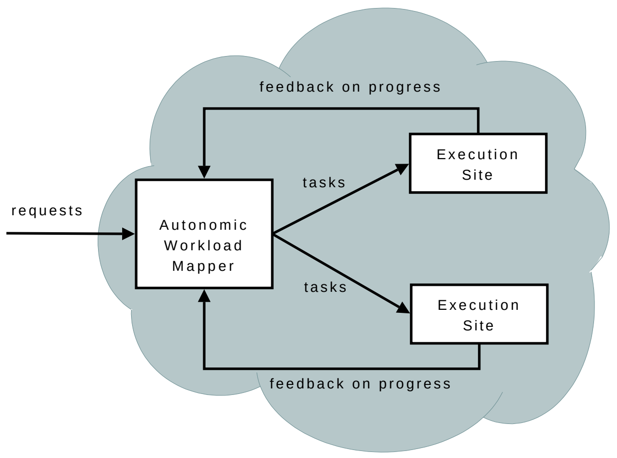

Figure 1: High level architecture.

In so doing, we explore functionalities that in future may be supported within a cloud, rather than by layering rich application functionality over lean cloud interfaces, as in Brantner *et al.* [4].

Wherever services are provided, service users have expectations; Service Level Agreements (SLAs) make explicit what expectations users can realistically place on a service provider [16], and may be associated with a charging model that determines the remuneration associated with certain Qualities of Service (QoS). Whether or not formal agreements are in place, decisions must nonetheless be made that influence the behaviors users experience, and service providers must put in place mechanisms that make such decisions.

In this paper, we assume the abstract architecture illustrated in Figure 1, where an *autonomic workload mapper* provides workload evaluation services implemented within a cloud. For the moment, we are non-specific about the nature of these services, but in due course details will be provided about support for workloads consisting of collections of queries or workflows. The *autonomic workload mapper* adaptively assigns tasks in the workload to execution sites. Given some objective, such as to minimize total execution times or, more generally, to optimize for some QoS target (whether these objectives are imposed by an SLA or not), the autonomic workload mapper must determine which tasks to assign to each of the available execution sites, revising the assignment during workload execution on the basis of feedback on the overall progress of the submitted requests.

In this paper, we investigate the use of *utility functions* [9] to make explicit the desirability of the state of a system at a point in time. In essence, a utility function maps each possible state of a system to a common scale; the scale may represent response times, numbers of QoS goals met, income based on some charging model for the requests, etc. In this setting, it is the goal of the *autonomic workload mapper* to explore the space of alternative mappings with a view to maximizing utility as measured by the utility function. We propose that utility functions, combined with optimization algorithms that seek to maximize utility for a workload given certain resources, may provide an effective paradigm for managing workload execution in cloud computing.

The remainder of this paper is structured as follows. Section 2 describes a methodology for developing utility based autonomic workload execution. Sections 3 and 4 describe the application of the methodology to workloads consisting of sets of workflows and queries, respectively. Section 5 presents some conclusions.

2 Utility Driven Workload Execution

When a utility-based approach is adopted, the following steps are followed by designers; instantiations of each of these steps are detailed for workloads consisting of workflows and queries in Sections 3 and 4, respectively.

Utility Property Selection: Identify the property that it would be desirable to maximize – useful utility measures may be cast in terms of response time, number of QoS targets met, profit, etc.

Utility Function Definition: Define a function $Utility(w, a)$ that computes the utility of an assignment a of tasks to execution sites for a workload w expressed in terms of the chosen property – for workload mapping, such a function can be expected to include expressions over variables v_e that describe the environment and the assignment a that characterizes the mapping for the components of w from abstract requests to tasks executing on specific execution sites.

Cost Model Development: Develop a cost model that predicts the performance of the workload given the information about the environment v_e and assignment a, taking into account the costs associated with adaptations.

Representation Design: Design a representation for the assignment a of workload components to computational resources, where adaptations to the assignment can be cast as modifications to this representation. For example, if a workload consists of a collection of tasks, then an assignment a of tasks to sites may be represented as a vector v where each element v_i represents task i, and each element value represents the execution site to which the task is assigned.

Optimization Algorithm Selection: Select an optimization algorithm that, given values for v_e, searches the space of possible assignments a with a view to maximizing the utility function; one benefit of the utility-based approach is that standard optimization algorithms can be used to explore the space of alternative mappings. Note that one benefit of the methodology is that it decouples the problem of meeting certain objectives under certain constraints into a modeling problem (i.e., to come up with a utility function) and an optimization problem (where standard mathematical techniques can be used).

Control Loop Implementation: Implement an autonomic controller [8] that: *monitors* the progress of the workload and/or properties of the environment of relevance to the utility function; *analyses* the monitored information to identify possible problems or opportunities for adaptation; *plans* an alternative workload execution strategy, by seeking to maximize $Utility(w, a)$ in the context of the monitored values for v_e; and *updates* the workload execution strategy where planning has identified an assignment that is predicted to increase utility.

Several researchers have reported the use of utility functions in autonomic computing, typically to support systems management tasks (e.g. [19, 3]); to the best of our understanding this is the first attempt to provide a methodology for the use of utility functions for adaptive workload execution.

3 Autonomic Workflow Execution

A cloud may host computational services in a specific domain; for example, the CARMEN e-Science cloud provides a collection of data and analysis services for neuroscience, and applications are constructed using workflow enactment engines hosted within the cloud [20]. In such a setting, autonomic workflow execution must determine how best to map workflows to the resources provided by the cloud.

3.1 Problem Statement

A workload w consists of a set of workflow instances i, each of which consists of a collection of tasks, $i.tasks$, and is evaluated through an allocation of tasks to a set of execution sites. The role of the autonomic workload mapper is to adaptively assign the tasks to specific sites.

3.2 Methodology Application

The methodology from Section 2 can be applied in this example as follows.

Utility Property Selection: Two utility properties are considered here, namely *response time* and *profit*. In practice, a single utility function is used by an autonomic workload mapper, but alternatives are shown here to illustrate how the approach can be applied to address different system goals.

Utility Function Definition: A utility function is defined for each of the properties under consideration. For *response time* we have:

$$Utility_w^{RT}(w, a) = 1/(\Sigma_{i \in w} PRT_w(i, a_i))$$

where, w is the set of workflows, a is a set of assignments for the workflows instances i in w, a_i is the assignment for workflow instance i, and PRT_w estimates the predicted response time of the workflow for the given assignment.

For *profit* we have:

$$Utility_w^{Profit}(w, a) = \Sigma_{i \in w}(Income(i, a_i) - EvaluationCost(i, a_i))$$

where $Income$ estimates the income that will be received as a result of evaluating i using allocation a_i, and $EvaluationCost(w, a)$ estimates the financial cost of the resources used to evaluate i. In this utility function, we assume that income is generated by evaluating workflows within a response time target, but that an $EvaluationCost$ is incurred for the use of the resources to evaluate the workflows. As the income depends on the number of QoS targets met, which in turn depends on reponse time, the definition of $Income$ is defined in terms of PRT_w. In cloud computing, the evaluation cost could reflect the fact that at times of low demand all requests can be evaluated using (inexpensive) resources within the cloud, but that at times of high demand it may be necessary to purchase (expensive) cycles from another cloud in order to meet QoS targets.

Cost Model Development: The cost model must implement $PRT_w(i, a_i)$; the predicted response time of a workflow depends on the predicted execution times of each of the tasks on their assigned execution site, the time taken to move data between execution sites, the other assignments of workflows in w, etc. The description of a complete cost model is beyond the scope of this paper, but cost models for workflows have been widely studied (e.g. [12, 18, 21]).

Representation Design: For each workflow instance $i \in w$, the assignment of the tasks $i.tasks$ can be represented by a vector v where each element v_i represents task i, and each element value represents the execution site to which the task is assigned.

Optimization Algorithm Selection: The optimization algorithm seeks to maximize $Utility(w, a)$ by exploring the space of alternative assignments a. As the assignments are represented as collections of categorical variables, each representing the assignment of a task to a specific execution site, an optimization algorithm must be chosen for searching such discrete spaces (e.g. [2]).

Control Loop Implementation: In autonomic workflow management [13], there is a requirement to halt an existing workflow, record information on the results produced to date, deploy the revised workflow in such a way that it can make use of results produced to date, and continue with the evaluation.

In practice, the utility functions described above prioritize different behaviors, and effective optimization can be expected to yield results that reflect those priorities. For example, $Utility_w^{RT}(w, a)$ will always seek the fastest available solution, even if this involves the use of costly computational resources. As a result, $Utility_w^{Profit}(w, a)$ will typically yield response times that are slower than those obtained by $Utility_w^{RT}(w, a)$, as it will only use expensive resources when these are predicted to give net benefits when considered together with the income they make possible. A detailed description of utility-based workflow execution in computational grids, including an experimental comparison of behaviors exhibited by different utility functions, is given by Lee *et al.* [12].

4 Autonomic Query Workload Execution

Early cloud data services are typically associated with fairly restrictive data access models with a view to enabling predictable behaviors, and do not provide full query evaluation [1]. However, more comprehensive data access services could provide access either to arbitrary query evaluation capabilities or to parameterized queries, thus giving rise to a requirement for query workload management, where collections of query evaluation requests can be managed by [11]: (i) an *admission controller*, which seeks to identify and disallow access to potentially problematic requests; (ii) a *query scheduler*, which determines when jobs are released from a queue for execution; and (iii) an *execution controller*, which determines the level of resource allocated to queries while they are executing. In this paper we discuss how utility functions can be used to direct the behavior of an *execution controller*. In comparison with recent work on workload management, a utility-driven approach can provide relatively fine-grained control over queries; for example, Krompass *et al.* [10] describe an execution controller in which the actions carried out at query runtime are job-level (i.e., reprioritize, kill and resubmit), whereas here the optimization makes global decisions (taking into account all the queries in the workload) that adaptively determine the resource allocations of individual queries on the basis of (fine-grained, collected per query) progress and load data.

4.1 Problem Statement

A workload w consists of a set of queries $q \in w$, each of which are evaluated on a collection of execution sites, potentially exploiting both partitioned and pipelined parallelism. Each query is associated with a distribution policy $dp(q)$, of the form $[v_1, v_2, \ldots, v_{|S|}]$, where $0 \leq v_i \leq 1$ and $(\Sigma_{i=1}^{|S|} v_i) \in \{0, 1\}$ where $|S|$ is the number of available execution sites. If the sum of v_i yields 1 then each v_i represents the fraction of the workload that is to be evaluated on the ith site using partitioned parallelism, and if the sum is 0 this represents the suspension of the plan. Where v_i is 0 for some i this represents the fact that execution site i is not being used for q. The role of the autonomic workload mapper in Figure 1 is to adaptively compute distribution policies for each of the queries in the workload.

4.2 Methodology Application

The methodology from Section 2 can be applied in this example as follows.

Utility Property Selection: Two utility properties are considered here, namely *response time* and *number of QoS targets met*. In the second case, we assume that each query is associated with a response time target.

Utility Function Definition: A utility function is defined for each of the properties under consideration. For *response time* we have:

$$Utility_q^{RT}(w, dp) = (1/\Sigma_{q \in w} PRT_q(q, dp(q)))$$

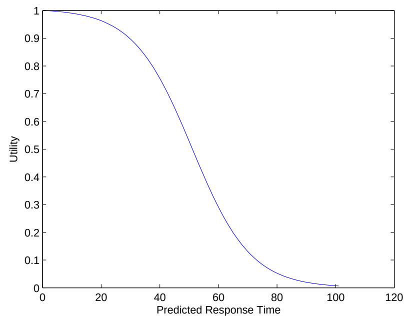

Figure 2: $QoSEstimate$ for a target response time of *50*.

where, w is the set of queries, dp is a distribution policy for the queries $q \in w$, and PRT_q estimates the predicted response time of the query for the given distribution policy.

For *quality of service* we have:

$$Utility_q^{QoS}(w, a) = \Sigma_{q \in w} QoSEstimate(q, dp(q))$$

where $QoSEstimate(q, dp(q))$ estimates the likelihood that the query will meet its QoS target using the given distribution policy from its predicted response time PRT_q. In practice, $QoSEstimate(q, dp(q))$ can be modeled using a curve such as that illustrated in Figure 2, which gives a score near to 1 for all queries estimated to be significantly within the target response time, and a score near to 0 for all queries estimated to take significantly longer than their target response time [3].

Cost Model Development: The cost model must implement $PRT_q(q, dp(q))$ for queries during their evaluation, and can build on results on query progress monitors (e.g. [5, 7]).

Representation Design: For each query $q \in w$, the distribution policy can be represented by a vector v where each element v_i represents the fraction of the work for q that is to be assigned to execution site i.

Optimization Algorithm Selection: The optimization algorithm seeks to maximize the utility function by exploring the space of distribution policies dp. As the assignments are represented as fractions, each representing the portion of the work to be assigned to a specific execution site, an optimization technique must be chosen for searching such spaces (e.g., sequential quadratic programming [6]).

Control Loop Implementation: The implementation of the control loop must be able to suspend an evaluating query, relocate operator state to reflect changes to the distribution policy, and continue evaluation using the updated plan. A full description of such a protocol is provided in the paper on Flux [17].

As an example of the behaviors exhibited by workflow execution management techniques, we have experimentally evaluated several such techniques using a simulator of a parallel query evaluation engine [15]. Figure 3 shows results for five different strategies: *No Adapt*, in which no runtime adaptation takes place; *Adapt 1* in which workloads are managed using action based control strategies (i.e. *if-then* rules based on Flux [17]) that seek to minimize response times by adapting whenever load imbalance is detected; *Adapt 2* in which utility functions are used to minimize response times, as in $Utility_q^{RT}$; *Adapt 3* in which *Adapt 2* is applied only when it is predicted that response time targets will be missed; and *Adapt 4* in which which utility functions are used to maximize the number of response time targets met, as in $Utility_q^{QoS}$. In this experiment, four queries each containing a single join are submitted at the same time to a cluster containing *12* execution sites, where one

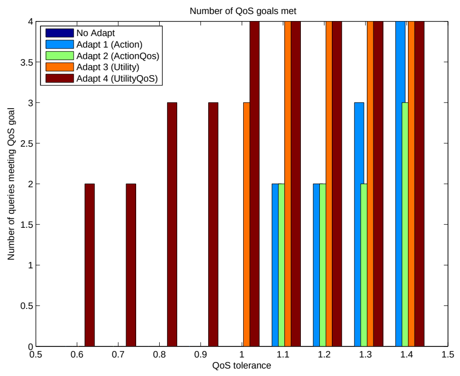

Figure 3: Numbers of Quality of Service Targets met by adaptive techniques [15].

of the sites is subject to periodic interference from other jobs broadly half of the time. In the experiment, the queries are associated with varying QoS targets (shown on the horizontal axis, with the more stringent targets to the left), and the number of queries meeting their reponse time targets is illustrated on the vertical axis.

The following can be observed: (i) Where no runtime workload execution adaptation takes place, no queries meet their QoS targets expressed in terms of response time. (ii) Queries managed by $Utility_q^{QoS}$ continue to meet (some) stringent QoS targets where the other methods fail – this is because optimization selectively discriminates against some queries where this is necessary to enable others to meet their targets. (iii) Queries managed by $Utility_q^{RT}$ meet more QoS targets than the action based strategies because the optimizer considers the combined costs or benefits of collections of adaptations in a way that is not considered by the action-based approaches. A broader and more detailed description of the approaches and associated experiments is provided by Paton *et al.* [15]. For the purposes of this paper, we note that optimization based on a utility function that aims to maximize the number of QoS targets met has been shown to out-perform action-based strategies and utility based strategies that target different goals, thus illustrating how utility based techniques can target application requirements.

5 Conclusion

This paper presents a utility-based approach for adaptive workload execution, and has illustrated its application to workloads consisting of workflows or queries. Recent research that explicitly focuses on data intensive cloud computing has addressed issues such as evaluation primitives (e.g. [14]) or the development of layered architectures (e.g. [4]). However, results from many different parts of the database community may usefully be revisited in the context of clouds; this paper considers workload management [11], and in particular the use of utility functions for coordinating workload execution. In this setting, a utility-based approach has been shown to be applicable to different types of workload, and utility-based techniques can be applied both to coordinate adaptations at different granularities and to address context-specific optimization goals. These context-specific goals allow utility functions to direct system behavior in a way that reflects the requirements of the contracts or SLAs that are likely to be prominent in cloud computing.

References

[1] R. Agrawal et al. The claremont report on database research. *ACM SIGMOD Record*, 37(3):9–19, 2008.

[2] C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. on Optimization*, 17(1):188–217, 2006.

[3] M.N. Bennani and D.A. Menasce. Resource allocation for autonomic data centres using analytic performance models. In *Proc. 2nd ICAC*, pages 229–240. IEEE Press, 2005.

[4] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska. Building a database on s3. In *SIGMOD Conference*, pages 251–264, 2008.

[5] S. Chaudhuri, V.R. Narasayya, and R. Ramamurthy. Estimating Progress of Long Running SQL Queries. In *Proc. SIGMOD*, pages 803–814, 2004.

[6] R. Fletcher. *Practical Methods of Optimization*. John Wiley&Sons, 1987.

[7] A. Gounaris, N.W. Paton, A.A.A. Fernandes, and R. Sakellariou. Self-monitoring query execution for adaptive query processing. *Data Knowl. Eng.*, 51(3):325–348, 2004.

[8] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[9] J.O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[10] S. Krompass, U. Dayal, H. A. Kuno, and A. Kemper. Dynamic workload management for very large data warehouses: Juggling feathers and bowling balls. In *VLDB*, pages 1105–1115, 2007.

[11] S. Krompass, A. Scholz, M.-Cezara Albutiu, H. A. Kuno, J. L. Wiener, U. Dayal, and A. Kemper. Quality of service-enabled management of database workloads. *IEEE Data Eng. Bull.*, 31(1):20–27, 2008.

[12] K. Lee, N.W. Paton, R. Sakellariou, and A.A.A. Fernandes. Utility Driven Adaptive Workflow Execution. In *Proc. 9th CCGrid*. IEEE Press, 2009.

[13] K. Lee, R. Sakellariou, N.W. Paton, and A.A.A. Fernandes. Workflow Adaptation as an Autonomic Computing Problem. In *Proc. 2nd Workshop on Workflows in Support of Large-Scale Science (WORKS 07), Proc. of HPDC & Co-Located Workshops*, pages 29–34. ACM Press, 2007.

[14] H. Liu and D. Orban. Gridbatch: Cloud computing for large-scale data-intensive batch applications. In *CCGRID*, pages 295–305. IEEE Computer Society, 2008.

[15] N.W. Paton, Marcelo A. T. de Aragão, and A.A.A. Fernandes. Utility-driven adaptive query workload execution. In *Submitted for Publication*, 2009.

[16] R. Sakellariou and V. Yarmolenko. Job scheduling on the grid: Towards sla-based scheduling. In L. Grandinetti, editor, *High Performance Computing and Grids in Action*, pages 207–222. IOS, 2008.

[17] M.A. Shah, J.M. Hellerstein, S.Chandrasekaran, and M.J. Franklin. Flux: An adaptive partitioning operator for continuous query systems. In *Proc. ICDE*, pages 353–364. IEEE Press, 2003.

[18] P. Shivam, S. Babu, and J. S. Chase. Active and accelerated learning of cost models for optimizing scientific applications. In *VLDB*, pages 535–546, 2006.

[19] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proc. ICAC*, pages 70–77. IEEE Press, 2004.

[20] P. Watson, P. Lord, F. Gibson, P. Periorellis, and G. Pitsilis. Cloud Computing for e-Science with CARMEN. In *2nd Iberian Grid Infrastructure Conference Proceedings*, pages 3–14, 2008.

[21] M. Wieczorek, A. Hoheisel, and P. Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25(3):237–256, 2009.