International Journal of Database Theory and Application
Vol.8, No.4 (2015), pp.207-214
http://dx.doi.org/10.14257/ijdta.2015.8.4.21

# Developing Self-adaptive Software System: A Case Study

Qingfeng Zhang, Jing Xu and Chao Zhang

*College of Computer and Control Engineering, Nankai University, Tianjin, China*
*zhangqf@mail.nankai.edu.cn, xujing@nankai.edu.cn, chaos_zc@163.com*

***Abstract***

*Current trends in software system, such as cloud and big data platform, are leading to rapid and continuing changes. At the same time, these systems will have to react to these changes at runtime to satisfy the potential Quality of Service (QoS). Self-adaptation is recognized as a practical way for a software system to meet QoS requirements. The Development of self-adaptive software is generally more challenging and more difficult due to their high complexity. To address these challenges, this paper reviews the related research of self-adaptive software system and reports a case study that investigates a self-adaptive concurrency controller for database system. Through the case we illustrate how to develop a self-adaptive software system. Compared with other traditional method, the experimental results demonstrate that our self-adaptive controller can effectively improve the database performance by adjusting the MPL value based on workload changes and QoS requirements. Finally some future trends in this area are prospected and discussed.*

***Keywords:*** *Self-adaptive, Software development, System architecture, QoS-Driven Developing, Case study.*

## 1. Introduction

With the rapid development of computer and network technology, the software system is facing rapid and continuing changes that may occur in the external environments and user requirements such as the changes in the system load, user preferences change, the change of the access protocol, resource attributes, etc. In traditional change process, system administrator may manually adjust the system parameters to ensure the quality of service (QoS). But this process manner cannot timely react to changes and continuous operation because of its inherent delay. So in order to ensure good QoS in terms of speed, complete time, cost, and many other measures, we need develop a self-adaptive software that are able to modify their behavior and system configurations at run-time to achieve certain objectives. The development of such systems has shown to be significantly more challenging than traditional software systems and become an important research topic in many diverse application areas. In order to study the development process and method of self-adaptive software systems, we designed a self-adaptive concurrency controller for database systems, through the controller to make the system achieve optimal performance by dynamically adjusting the Multiprogramming Level (MPL) in Database.

The rest of the paper is organized as follows. In section 2 we examine some related works. In Section 3 we introduce some background of our case study and describe our development process of the self-adaptive MPL controller. After that we provide experimental evaluation of our work in a simulated environment on MySQL database in Section 4. Finally section 5 we conclude the paper with some planned future work.

## 2. Related Works

### 2.1. Self-adaptive Software System

In recent years, great research efforts have been made in self-adaptive software system. The goal of self-adaptive software is to alleviate the management problem of complex software systems that operate in highly changing and evolving environments. Such systems should be able to dynamically adapt themselves to their environment with little or no human intervention in order to meet both functional requirements concerning the overall logic to be implemented and nonfunctional requirements concerning the quality of service (QoS) levels that should be guaranteed [1].

It has been widely recognized that the architecture of self-adaptive software systems should include one or more control loops to perform self-adaptation tasks [2]. As a result, some interesting design patterns [3], high-level reference models [4] and frameworks [5-6] have been developed.

A reference model for the architecture of a self-adaptive software system has been presented in [3]. This paper suggests architecting the system along three different layers that interact with each other by reporting status information to the above layer and issuing adaptation directives to the layer below.

### 2.2. QoS-Driven Adaptation

Recently, QoS issues have obtained great interest in the software engineering research community. A lot of different approaches have been followed so far, spanning the use of QoS ontologies, the definition of QoS-aware framework [7], and the QoS-Driven adaptation middleware [8] of distributed system.

In general, all these approaches to automatic service composition or QoS-aware framework only consider one single aspect of the QoS requirements, which not only has to search for work plans for a given request, but also needs to guarantee global QoS requirements. This will bring about new challenges. In this paper we will select some QoS metrics to monitor and these metrics will be used as the inputs of our self-adaptive controller.

### 2.3. Adjusting MPL

Many recent research looks at the problem of automatically tuning the multiprogramming level to improve the server performance under varying workloads. MPL is the maximum number of requests that can be processed concurrently by the server. Many admission control approaches are essentially methods for tuning the MPL.

In [9] the authors investigate how to effectively schedule requests, rather than how to choose an optimal MPL. Finding a good MPL is simply a requirement for being able to perform good scheduling. They then use a feedback controller to dynamically adjust the MPL value. The paper mentions that one of the goals of the investigation is to set the MPL low enough that there is room for the schedule to show differentiation in effectiveness, since there can be a range of MPLs which give similar throughput. Their ideas are different from other researchers.

Mohammed Abouzour and others in [10] presented a controller that can adjust the MPL of a database server by monitoring throughput level at the server. But in [11] the authors conduct a series of experiments to find the best MPL value for their experiment environment.

In our case study, we develop a self-adaptive MPL controller depending on the workload changes and QoS requirements. Experimental results show that our self-adaptive controller can effectively improve system performance by tuning the MPL value of database server.

## 3. Case Study

In this section, we describe our experience of case study [12] of developing a self-adaptive system.

### 3.1. Requirement Description

In the case study, we focus on the problem of automatically tuning the multiprogramming level to improve database server performance under varying workloads. The MPL is one of the most important configuration parameters of the database server. The MPL determines how many requests can be allowed to execute concurrently. Generally, The MPL of the database is fixed, but through experiment research we know that for different database workloads the constant MPL may have bad impacts on system performance and may lead to some QoS violations. Seting a high MPL value may cause more contention on shared system resources and even lead to deadlock, system crashes and other problems of overloading. While if he MPL is too low, the system resources will not be fully utilized and the system throughput will be degraded. So in order to timely react to workload changes, we need to develop a self-adaptive middleware to automatically adjust the MPL value.

Specifically, our work can be described as follows: "Given a set of queries Q1, Q2, Q3 ... Qn, concurrently executing on the same database server. It is assumed that the arrival sequence is unchanged. Please develop a self-adaptive controller to adjust the MPL value depending on the workload changes to achieve the optimal system performance."

### 3.2. Workload-Aware System Architecture

In this section, we present a workload-aware architecture of our self-adaptive MPL controller which is depicted in Figure 1.

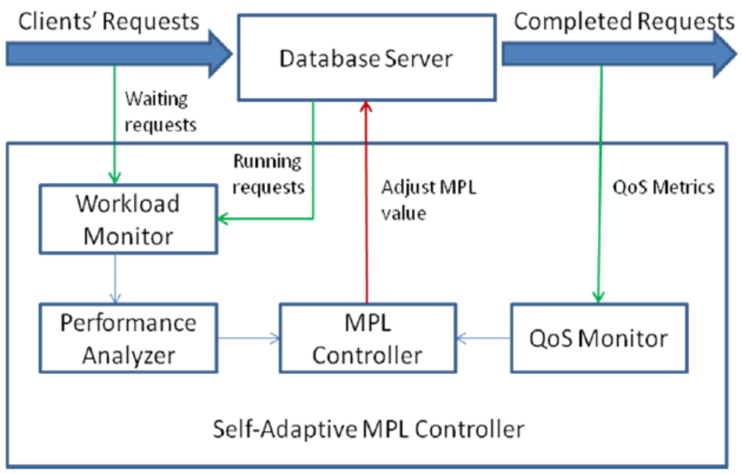

**Figure 1. Workload-aware MPL Controller**

Figure 1 shows that the MPL controller has multi inputs and single output and is composed of four components. In our architecture, we assume that the control interval is fixed. But in future we will plan to explore the control strategy with not fixed control interval. In that case, we will examine if the QoS targets are violated to determine whether the MPL parameter should be reconfiguration.

In what follows, we identify and specify the role of each component of our self-adaptive architecture.

**a) Workload Monitor**

Workload Monitor is designed to track workload changes. These changes include the number and the type of all user requests. The total number of requests is the sum of the number of waiting requests and the number of running requests. The number and the type of user requests will be used as input parameters of our controller.

**b) QoS Monitor**

This is another component to monitor database QoS metrics. Commonly database system has many QoS metrics such as latency, throughput, resource utilization. But in our controller we focus on two metrics. One is the total completion time of all the query tasks and another is the throughput of database server. These monitoring information will help us decide how to adjust the MPL value.

**c) Performance Analyzer**

Performance Analyzer mainly deals with the data and information collected by the workload monitor to do some statistical analysis and performance prediction. In addition, we will construct the performance model based on the interaction among concurrent tasks.

**d) MPL Controller**

This is the component that executes the adaption actions to adjust the database MPL value. After receiving the reconfiguration command, the MPL controller will select the best configuration to meet the QoS requirements considering each recommended value and dependencies. The MPL value selected by the controller will be used during the next control interval.

In the system design process, we advocate using black-box techniques to implement the control method. The black-box approach makes us more convenient to adjust the MPL value outside the database server.

## 3.3. System Implementation

In this section, we discuss several implementation issues and challenges for implementing the MPL Controller.

As a self-adaptive MPL controller, the most important thought of the implementation is external controlling. So we treat the database server as a black box and leave all other tuning parameters of the database at their default value. In addition, we do not analyze the internal behavior of the database system.

In order to get a good portability and scalability, the system implementation takes the following measures:

- In order to achieve good portability, all the program of the structures and algorithms should be realized by the classic implementation of the C# language and .Net Framework4.0.
- Using the component-based development method to enhance the flexibility and scalability of the software system. Furthermore, the component-based method can enable software easier to reuse. For example, the operation of connecting to the database system is designed as a plug-in component. Our system can connect to different database systems conveniently without modifying too much code.

**a) Performance Prediction**

In order to dynamically adjust the system MPL value, we need to construct models to capture the workload changes and predict the system performance to recommend the optimal parameter. Recently some works clearly show that in a database workload the query type has a big impact on the system performance. So we will build our performance model based on query TYPE. When we adjust the MPL value and new query will be added in the running pool, we will predict the query run time using the performance model. Our performance model is built by executing the each query type in an off-line learning phase. We use the linear regression algorithm in the WEKA toolkit [13] to drive our model from the experimental data.

We predict the execution time of query type i as:

$$Esti = ci * M + Ci$$

In the above equation, M is the number of queries currently executing with query i and Ci estimates the runtime of query i while run alone. The coefficient ci represents the amount of time each query in the system adds to the execution time of a query of type i.

If the predicting time exceeds the timeout threshold, we will know that that the server is overloaded and the OoS targets may be violated, so the query should be rejected. Otherwise, the query will be executed.

**b) Control Algorithm**

Adaptive control is a continuous optimization process, there are many adaptive control algorithms such as hill-climbing algorithm, Parabola Approximation algorithm, genetic algorithm. In our case, we select the hill-climbing algorithm to adjust the MPL value.

Hill-climbing algorithm is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

In our case, our self-adaptive MPL control algorithm can be described as follows:

**Step 1:** Firstly, we should determine the optimal system goal. Here our algorithm goal is to keep the maximum throughput and avoid the timeout of each query.

**Step 2:** Start with an initial MPL value. We can user the performance model to predict the system performance and select a starting MPL value.

**Step 3:** At the end of the current control interval, we select a recommend MPL value according to the workload changes, and using the performance analyzer to predict the QoS level.

**Step 4:** If the QoS level is improved, we believe that the adjustment is a good move, and then set the new MPL value as the current point and repeat the Step 3. Otherwise if we do not benefit from the adjustment in either the forward or backward direction, terminate and keep the current MPL value until the next control interval.

The above shows the execution progress of the hill-climbing algorithm. It is relatively simple to implement, making it a popular first choice. Of course there are many advance algorithms may give better results, we also plan to experiment with more advance machine learning techniques for adjusting the MPL value in database system.

## 4. Evaluation

In this section, we evaluate our self-adaptive MPL controller from several aspects based on our case study.

Our controller is developed by a component-based way and the architecture is strictly interface-based, so it supports most of the runtime adaptations such as binding database connection and system parameters reconfiguration. In order to evaluate the effectiveness, we conduct some experimental studies to verify the practicality and efficiency of our self-adaptive MPL controller.

### 4.1. Set up

Our experiments were conducted on a machine with an Intel i5 3.1GHZ processors and 8GB of RAM. The machine run on Windows Server 2008 and the database system is version 5.5.25 of MYSQL. The buffer pool size of the database was set to 2.4G.

### 4.2. DateSet and Workload

We use TPC-H benchmark to do our experiments [14]. The standard TPC-H benchmark provides two programs for generating data and queries, DBGEN and QGEN. We use these programs to generate our datasets and query workload.

**DataSet:** We use the DBGEN tool to generate the TPC-H database with scale factors of 10GB.

**Query Workload:** Using the QGEN tool to generate our query workloads. We elect the 10 moderate weight query templates and generate 10 instances for each query template as required by the TPC-H specification. That is to say, the completion time of the selected query templates is medium for all of the TPC-H types. Specifically, our workloads are comprised of TPC-H queries 2, 3, 6, 7, 8, 9, 10, 14, 19 and 21 on the TPC-H database.

### 4.3. Experiments

Firstly, we use a workload consisting of 10 instances of each TPC-H query template, for a total of 100 queries. We assume that the arrival sequence of each query instance is unchanged. Next we execute the workload in two different ways: One is the fixed MPL value (M= 20), while in another case the MPL value is adjusted by a self-adaptive controller. In each case, we record the total completion time of the whole workload and the average time of each query template.

Figure 2 shows a comparison in total completion time between the fixed MPL value and the self-adaptive implementations.

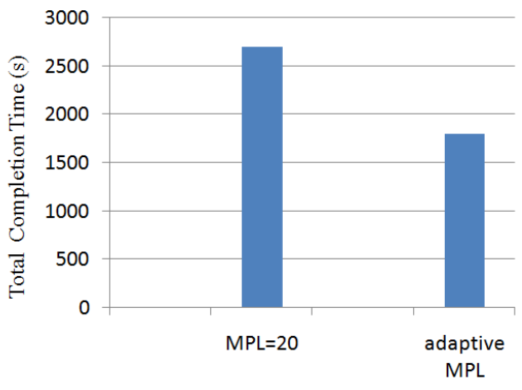

**Figure 2. Total Completion Time in Two different Ways**

From the Figure 2 we know that adaptive MPL method can significantly improve the system performance than the fixed MPL strategy. Moreover, theses improvement just because we adjust the MPL value in a self-adaptive way.

Next we will examine the execution time of single query template as Figure 3.

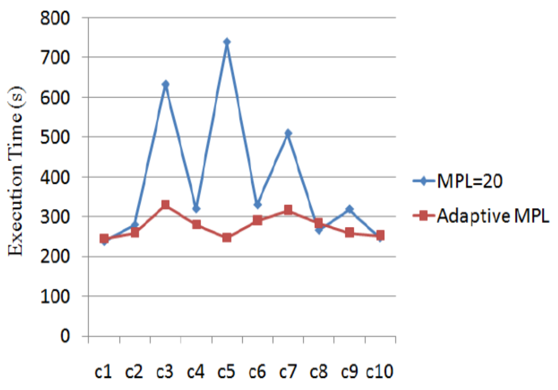

**Figure 3. Execution Time of 10 Instances of Q3 Template in Two different Ways**

The Figure 3 demonstrate that the execution time of query task will severely changed with the fixed MPL (M=20). But with a self-adaptive controller, the execution time will change smoothly.

In brief, our self-adaptive MPL controller can adjust MPL value automatically according to the workload changes and using self-adaptive strategy can lead to a better performance than traditional method.

## 5. Conclusions and Future Work

In this paper, we present a case study on developing self-adaptive software system. In order to develop a self-adaptive MPL controller of database server, we build a component-based self-adaptive software architecture. In the case study, we illustrate the implementation of each component of our self-adaptive controller. Then we evaluate the strengths of our adaptive controller through some experimental studies. We conducted these experiments using TPC-H benchmark on MySQL database. Our case study shows that component-based architecture is effective for implementing self-adaptive systems. The obtained results are encouraging and show the effectiveness of our self-adaptive MPL controller to improve system performance.

There are several open directions for future work in the area of developing self-adaptive software system. Actually we are planning to use more advance machine learning methods to build the performance prediction model of concurrent workloads [15]. Another interesting future direction is to explore the dynamic adaptive control algorithm to realize the adaptive process. In addition, we believe that there also have some important challenges that we must face in developing self-adaptive software systems. But in other hand we know that the self-adaptive software is also an opportunity issue and we are beginning to extend this work to solve more challenging problem such as workload management and resource allocation.

## Acknowledgements

This work is supported by Tianjin Science and Technology Committee (No. 12JCZDJC20800).

## References

[1] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. LoPresti and R. Mirandola, “Moses: A framework for qos driven runtime adaptation of service-oriented systems”, IEEE Transactions on Software Engineering, **(2011)**.

[2] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee and R. deLemos, "Software engineering for self-adaptive systems: A research road map", Proceedings of the Dagstuhl Seminar in the Software Engineering for Self-Adaptive Systems, **(2008)**; Dagstuhl, Germany.
[3] A. J. Ramirez and B. H. C. Cheng, "Design Patterns for Developing Dynamically Adaptive Systems", Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, **(2010)** ; New York, NY, USA.
[4] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software", Proceedings of the IEEE International Conference on Software Engineering, **(2010)**; Shanghai, China.
[5] Y. Wu, Y. Wu, X. Peng and W. Zhao, "Implementing Self-Adaptive Software Architecture by Reflective Component Model and Dynamic AOP: A Case Study", Proceedings of the 10th International Conference on Quality Software, **(2010)**; Zhangjiajie, China.
[6] A. Elkhodary, N. Esfahani and S. Malek, "FUSION: a framework for engineering self-tuning self-adaptive software systems", Proceedings of the 18th ACM SIGSOFT international symposium, **(2010)**; Santa Fe, NM, USA.
[7] W. Kang, S. Son and J. Stankovic, "Design, implementation, and evaluation of a qos-aware real-time embedded database", IEEE Transactions on Computers, **(2010)**.
[8] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti and R.Mirandola, "QoS-Driven Runtime Adaptation of Service Oriented Architectures", Proceedings of Seventh Joint Meeting of the European Software and the ACM SIGSOFT Symposium, **(2010)**; Santa Fe, NM, USA.
[9] B. Schroeder, M. Harchol-Balter and A. Iyengar, "How to determine a good multi-programming level for external scheduling", Proceedings of ICDE, **(2010)**; Long Beach, California, USA.
[10] M. Abouzour, K. Salem and P. Bumbulis, "Automatic tuning of the multiprogramming level in Sybase SQL Anywhere", In Workshop on Self-managing Database Systems (SMDB), **(2010)**; Long Beach, California, USA.
[11] S. Tozer, T. Brecht and A. Aboulnaga, Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads, Proceedings of ICDE, **(2010)**; Long Beach, California, USA.
[12] R. K. Yin, "Case Study Research: Design and Methods", 4th edn. Sage publications, Thousand Oaks, **(2009)**.
[13] WEKA workbench. http://www.cs.waikato.ac.nz/ ml/weka/.
[14] TPC-H benchmark specification, http://www.tpc.org/tpch/.
[15] D. Rughetti, P. D. Sanzo, B. Ciciani and F. Quaglia, "Machine learning-based self-adjusting concurrency in software transactional memory systems", Proceedings of MASCOTS, **(2012)**; Washington, DC, USA.

## Authors

**Qingfeng Zhang**, he was born in 1979. Now he is a Ph.D. candidate at Nankai University. His research interests include software engineering, data analysis and performance evaluating technology.

**Jing Xu**, she was born in 1967. She received her Ph.D. degree from Nankai University in 2003. Now she is a professor at Nankai University, and the member of CCF. Her research interests include software engineering, software testing and information technology security evaluation.

**Chao Zhang**, he was born in 1991. Now he is a master candidate at Nankai University. His research interests include software engineering, software testing and data analysis technology.