Experience on Building an Architecture Level Adaptable System

Xu Zhang and Chung-Horng Lung

Department of Systems and Computer Engineering
Carleton University, Ottawa, Ontario, Canada
{zhangxu,chlung}@sce.carleton.ca

Abstract. Distributed and concurrent systems have become common in enterprises, and the complexity of these systems has increased dramatically. The self-adaptive feature can be advantageous for complex systems, because it can acclimate to a dynamically changing environment. To achieve this goal, this paper presents a Self-Adaptive Framework for Concurrency Architecture (SAFCA). SAFCA includes multiple concurrency architectural alternatives and is able to adapt to an appropriate architecture based on changes in the environment and the control policy. With an autonomic control, SAFCA can handle bursty workloads by invoking another architectural alternative at runtime instead of statically configured to accommodate the peak demands, which requires higher system resources even when they are not needed. Experimental results demonstrate that SAFCA can improve performance. The experience can be useful for building complicated systems that have multiple configurations or diverse demands, such as cloud computing.

Keywords: software adaptation, software architecture, software performance engineering, concurrency patterns.

1 Introduction

The average complexity of computer systems and the number of computing devices in use have been increasing dramatically [3]. As a result, IT personnel have to shoulder the burden of time-consuming (and therefore expensive) supporting tasks such as configuration, maintenance and system performance evaluation [5]. Further, manual control of a large distributed computing system is invariably prone to errors.

The goal of autonomic computing, initiated by IBM in 2001 [3], is to define rules for a system to control its behavior so that the system regulates its actions to automatically configure, heal, protect, and optimize itself [5]. More research efforts and IT companies have launched research projects related to autonomic computing recently [7] including in the area of distributed and concurrent applications, due to their high complexity.

Furthermore, the Internet is known for its dynamic nature. Under normal circumstances, the growing server farm is more than adequate to handle regular traffic demands. However, during special events, the server farm may become unavailable

M. Bajec and J. Eder (Eds.): CAiSE 2012 Workshops, LNBIP 112, pp. 171–179, 2012.

due to unprecedented demands [10]. On the other hand, many systems today are configured initially and statically to handle worst-case scenarios; in other words, systems are over-provisioned. The problem with this approach is that many resources will be wasted in normal condition scenarios.

Another challenge in software architecture is to conduct evaluation for the performance perspective, as there are usually uncertainties and lack of concrete data early in the life cycle. The problem could become more difficult if multiple software architectural alternatives exist.

This paper presents a self-adaptive system, self-adaptive framework for concurrency architectures (SAFCA), that supports software adaptation at the architecture level for the distributed and concurrent problem domain. The self-adaptive system consists of multiple software architectural alternatives. The system can adapt to changing demands at runtime by using the appropriate alternative. The main objective of the adaptation is to better utilize system resources and increase performance.

This paper is organized as follows: Section 2 provides the background information on distributed and concurrency architectures. Section 3 describes the self-adaptive framework, SAFCA. Section 4 presents experiments and the performance results. Finally, Section 5 is the conclusion.

2 Background

This section provides brief background information on the primary concurrency architectural patterns used in the paper.

2.1 Concurrency Architectural Patterns

Distributed and concurrent programming has been widely used in many applications. There are several known concurrency architectural alternatives: single-thread, dynamic thread Creation (DTC), Half-Sync/Half-Async (HS/HA), and Leader/ Followers (LFs) [8]. Both HS/HA and LFs are thread pool-based approaches. According to our previous performance modeling studies using the layered queuing networks [11], HS/HA has higher efficiency than LFs for high demands. Therefore, HS/HA is chosen as the main thread pool-based architecture used in the SAFCA.

DTC, on the other hand, is selected for the study, because it is the simplest and the common design that does not have a thread pool. Our framework actually supports LFs as well. But the LFs pattern is not included in this comparative evaluation.

Dynamic-thread-creation (DTC): The DTC architecture is commonly used in multi-threaded programming, especially for server applications. DTC is based on the idea of a thread-per-request. DTC creates a thread for each new request. A thread dies after processing its request [9]. DTC is relatively easy to program. However, a thread has to be dynamically created per request, there is an overhead associated with it.

Half-Sync/Half-Async (HS/HA): In order to simplify programming without unduly reducing performance, HS/HA decouples asynchronous and synchronous service processing in concurrent systems [8].

External input first arrives at the asynchronous layer. After the asynchronous layer processes the input, the request is stored in a queue in the queuing layer. The function of the queuing layer is to buffer input from the asynchronous layer to the synchronous layer and inform the synchronous layer that input is now available. Worker threads in the synchronous layer retrieve input from the queue and process the input further. Layers are independent from each other and can perform operations concurrently.

2.2 Self-adaptive Systems

The aim of self-adaptive or autonomic computing is to provide a solution that runs services with minimum or lower cost, capable of both scaling up and scaling down the system resources responding to dynamic demands. This requires system elasticity, in terms of allocating or using resources as they are needed. Therefore, systems must adapt to changes in the environment quickly and since manual server reconfiguration has been shown to be inadequate [4], self-adaptive solutions are better suited for such problems. Autonomic computing has received more attention recently in software engineering, e.g., [1][2][6], mainly due to the increased complexity of systems. One crucial research issue is to build the capability into the system to adapt its behavior in response to the dynamically changing environment.

In [12], we also demonstrated the feasibility of SAFCA using a simple approach based on queue length. The idea of the queue length-based approach is simple, i.e., if the threshold is passed, adaptation will be invoked. But it is challenging to determine queue length and oscillations may happen. In this paper, we devise a policy to detect the traffic burst which triggers software adaptation.

3 Self-adaptive Framework for Concurrency Architectures

SAFCA is designed to support adaptation at the architecture level during runtime based. In other words, adaptation occurs from one architectural alternative to another. This section first describes the main concept of the approach. The framework and the main components are then presented. After that, the self-adaptation scheme is described. The monitoring mechanism of busty traffic and the policy of managing busty traffic autonomically will be demonstrated.

3.1 Problem Scope and Main Concept

The paper considers three conditions: 1) when an overloading request burst occurs, the framework can achieve acceptable response time and decrease the message or packet loss ratio; 2) the software resource (for example, the number of threads) utilization under normal load condition can be minimized; 3) the self-adaptive framework must be practical in the sense that it can be easily implemented.

HS/HA works well under normal workload conditions. However, if a burst occurs, the pre-configured size of the thread pool becomes the performance bottleneck for HS/HA. On the other hand, DTC can create a large number of threads to handle the sudden burst of requests. But the overhead of thread creation and destruction in DTC

makes it inferior, compared to HS/HA, under normal workload condition. For this reason, the self-adaptive system containing both HS/HA and DTC architectures is built to validate the concept of self-adaptation. In normal conditions, HS/HA is used. If a burst arrives (based on the burst detection policy), DTC will be activated immediately, because HS/HA has a pre-configured pool size. During this period, both HS/HA and DTC alternatives are running concurrently or simultaneously (for a multi-core system). The approach is compared against a system running either HS/HA or DTC alone for performance evaluation.

3.2 Overview of the Framework

In order to support the tasks described in Section 3.1, Figure 1 depicts the framework and the four main components. The *Monitor* component gathers information regarding the queue length, response time, arrival rate, and number of threads currently running (number of threads that have received a request but has not sent a reply). The *Decider* then computes statistical-average-quantities, such as average queue length, average response time, etc. Based on collected and calculated information, the *Decider* decides if any action should be taken. For example, if the queue length exceeds a threshold, the *Decider* notifies the *Executor* component that the current workload condition is heavy. The *Executor* will then instruct the *Architecture Manager* to put new requests into the appropriate queue.

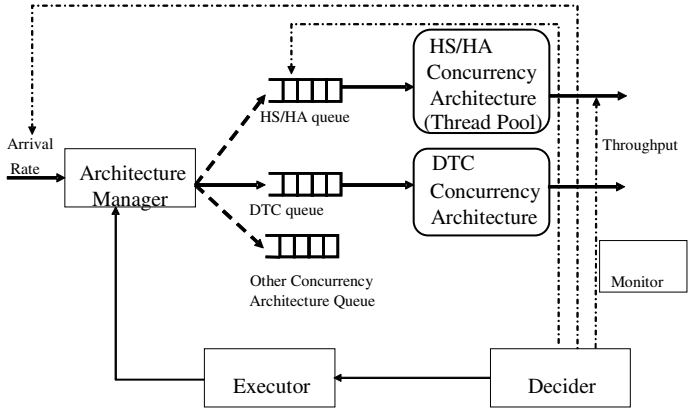


Fig. 1. Self-Adaptive Framework for Concurrency Architectures Overview

The framework consists of multiple architectural alternatives. Currently, three alternatives are included: DTC, HS/HA, and LFs (not shown in Figure 1 and not included in performance evaluation). Each architecture alternative has its own queue.

The *Architecture Manager* has a high scheduling priority so it can respond to incoming requests immediately. Once the destination queue becomes full, the *Architecture Manager* drops any new requests.

3.3 Self-Adaptation Policy for Bursty Traffic

The self-adaptation policy is designed so that HS/HA is the default architecture. During normal scenarios, DTC is not active and the *Architecture Manager* puts new requests in the *HS/HA queue*. The trigger to self-adaptation is based on arrival rate and average response time, see Section 3.4.

When an arrival burst occurs and the response time also increases over a certain level, the *Architecture Manager* begins to place new requests to *DTC queue* to deal with bursy demands. When the burst is over and the arrival rate returns to the pre-burst level, the *Architecture Manager* puts new requests to the original *HS/HA queue*.

3.4 Burst Detection Policy

Monitoring and measurements are two important elements for autonomic computing. Monitoring and measurements are used for burst detection in our approach. Detection of a burst could range from a simple threshold-based approach according to the queue length [12] to a sophisticated method. This paper presents an approach based on both the arrival rate and the average response time for message processing. Arrival rate alone cannot indicate if the system has reached its capacity. Therefore, the average response time is also used.

In the burst detection policy, the standard deviation (σ) of previous arrival rates and the mean arrivals in a pre-configured interval are used.

Assume *r1, r2, r3,..., rn* are arrival rates for sampling intervals *1, 2, 3,..., n,* respectively. If the difference between the current arrival rate and the mean arrival rate is greater than the standard deviation of previous arrival rates (σ), then a burst is assumed to have occurred. However, this does not mean that the system is overloaded, but a self-adaptive action needs to be taken. If the difference between the current response time and previous response time also increases by *y%* (a pre-configurable parameter) of the previous response time, then the *Decider* notifies the *Executor* to take actions and the mean arrival rate is reset to zero. From this point on, a new mean arrival rate and a new standard deviation will be calculated.

If the current arrival rate is less than the mean arrival rate, and their difference is greater than the standard deviation of the arrival rates (σ), then the burst is assumed to be over. The *Decider* notifies the *Executor* to free resources and mean arrival rate is reset to zero. From this point on, a new mean arrival rate and a new standard deviation will be calculated.

4 Experiments and Analysis

This section presents experiments conducted and results. The performance of SAFCA is compared with that of standalone HS/HA and DTC without adaptive control.

4.1 Experiment Settings

The experiments consider a multi-tier system. The server receives message or requests from multiple clients, and the traffic generated by those clients contains

random bursts. Each request received by the server is processed including CPU-bound operations and I/O-bound operations, and then a reply is sent back to the client.

Our test bed consists of one server machine (3.0 GHz Pentium 4 systems with 3.49 GB of RAM) and a client machine (3.0 GHz Pentium 4 systems with 3.49 GB of RAM) connected to a Phoebe Ethernet Switch (8-Port 10/100Mbps Auto/MDIX). SAFCA is developed with SUN JDK 1.6 as the Java platform running on Microsoft Windows XP Professional on the server side. Multiple clients generate traffic and send requests to the server. The client traffic generator is also developed with the same platform as that of the server. A number of experiments have been conducted. TABLE I lists the parameters used for the experiments.

Table 1. Parameters used for experiments

Experiment Parameters	Value
burstAverageArrivalRate	200 messages/sec
normalAverageArrivalRate	50 messages/sec
maxBurstDuration	20 internvals
minBurstDuration	10 intervals
maxNormalDuration	30 intervals
minNormalDuration	20 intervals
sampleIntervalLength	5 sec
queueSize (HS/HA)	25
queueSize (DTC)	25
threadPoolSize	60

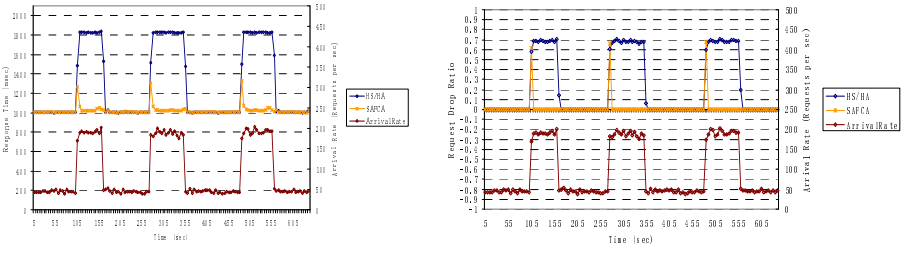
4.2 Performance Evaluation of SAFCA

This section evaluates the performance of SAFCA with HS/HA and DTC in terms of response time, request drop ratio, and utilization. As described in Section 3.3, the policy of SAFCA is to initially send requests to the HS/HA queue. When the arrival rate has increased by more than one standard deviation from the previous average arrival rate and the response time also has increased by more than 20% (a configurable value), SAFCA sends new requests to the DTC queue. If the arrival rate has decreased by more than one standard deviation, SAFCA sends new requests back to the HS/HA queue. The results show that SAFCA offers better performance.

SAFCA and Standalone HS/HA

Figure 2(a) depicts the response times for SAFCA and standalone HS/HA on the primary y-axis (on the left) and the arrival rate on the secondary y-axis (on the right). The response time of SAFCA is low during normal workload because it uses HS/HA which is more efficient for normal workload. During bursts, the response time is still low because SAFCA dynamically invokes DTC to cope with the bursts. This arrangement is efficient because with HS/HA, the configured number of threads (60) is sufficient during non-burst periods and no new threads are created. On the other hand, DTC can accommodate high demands during burst periods because it can create more threads. For standalone HS/HA, the throughput bottleneck is due to its fixed number of threads.

The spikes of the response time are primarily due to bursts and the monitoring interval. If the interval is reduced, the spikes can be shortened.



(a) (b)

Fig. 2. Comparison of Response Time (a) and Drop Ratios (b) for SAFCA and HS/HA

Figure 2(b) illustrates the request drop ratio of for SAFCA and HS/HA. The drop ratio of SAFCA is close to 0 (except when a burst first starts) during both normal workload and burst workload. Again, the drop ratio of SAFCA for the initial burst periods could be reduced by shortening the monitoring or sampling interval.

Figure 3 compares the thread utilization for SAFCA and standalone HS/HA. The thread utilization is always low (between 10% and 20%) for HS/HA because the thread pool size is fixed. However, because SAFCA uses DTC to create more threads when needed during bursts, the resource is better utilized with utilization mostly in the range of 60% to 70%.

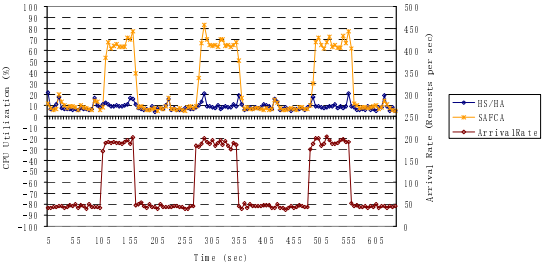


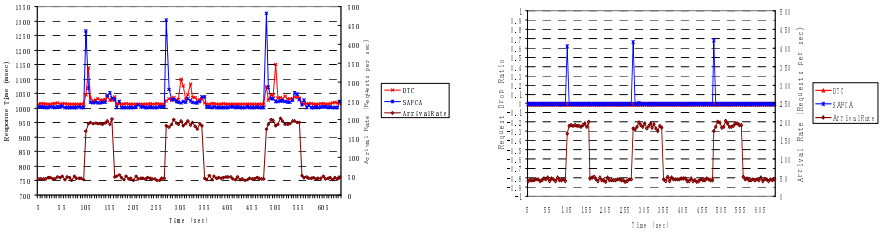
Fig. 3. Comparison of CPU Utilization for SAFCA and HS/HA

SAFCA and Standalone DTC

In this experiment, the response time, the request drop ratio, the thread utilization, and the number of created threads are measured for performance evaluation. The results show that SAFAC has better performance in most cases.

Figure 4(a) shows the response times for SAFCA and DTC. Except when the burst first started, SAFCA has a better response time than that of DTC in most cases. Again, the sharp spikes of SAFCA can be mitigated using shorter monitoring length.

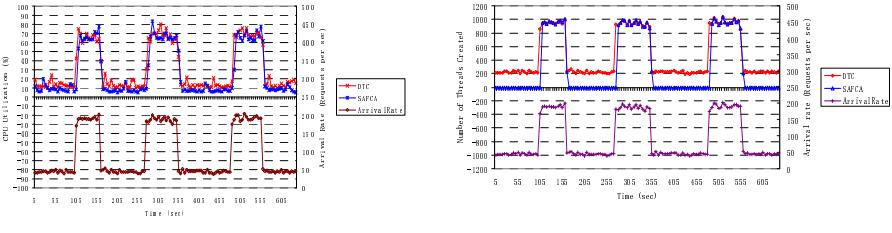
Figure 4(b) presents the drop ratio of for SAFCA and DTC. Both SAFCA (except when the burst first starts) and DTC have a loss ratio close to 0.



(a) (b)

Fig. 4. Comparison of Response Time (a) and Drop Ratios (b) for SAFCA and DTC

Figure 5(a) illustrates that SAFCA has similar thread utilization as that of DTC in both normal workload and burst workload conditions.In terms of resource usage, Figure 5(b) shows that DTC creates more threads and consumes more resources than SAFCA during normal workload condition. Each thread created requires memory space, CPU cycles, and the operating system overhead for thread creation/destruction. As depicted in Figure 5(b) in one sampling interval, DTC creates about 200 new threads even under normal conditions. In comparison, no additional threads are created dyanmically or only 60 existing threads in the thread pool of SAFCA can handle the normal workload. Since normal workload periods are typically much longer than burst periods, SAFCA is more resource efficient than DTC.



(a) (b)

Fig. 5. Comparison of CPU Utilization (a) and Number of Threads Created (b) for SAFCA and DTC

5 Conclusions

In order to effectively utilize resources under dynamic workloads, a self-adaptive framework, SAFCA, was proposed and developed. According to the results obtained from a number of experiments, SAFCA improved the performance under various workloads through an adaptive mechanism. In comparison to the standalone HS/HA and DTC, SAFCA exhibited performance gains without the need of over-provisioning as often adopted for thread pool-based approach. Under normal workload conditions, SAFCA has a better resource usage than DTC-only system. The concept could be useful for other applications to support scaling up and down of a system or cloud computing where many configurations and diverse resources and demands exist.

References

[1] Brun, Y., Di Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., Shaw, M.: Engineering Self-Adaptive Systems through Feedback Loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Self-Adaptive Systems. LNCS, vol. 5525, pp. 48–70. Springer, Heidelberg (2009)

[2] International Symp. on Software Engineering for Adaptive and Self-Managing Systems (2006-2012)

[3] Enterprise Management Associates, Practical Autonomic Computing: Roadmap to Self-Managing Technology. Tech. Rep., IBM, Boulder, CO (January 2006)

[4] IBM Autonomic Computing Architecture Team, An Architectural Blueprint for Autonomic Computing, Tech. Rep., IBM, Hawthorne, NY (June 2006)

[5] Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computer 36(1), 41–52 (2003)

[6] Kramer, J., Magee, J.: Self-Managed Systems: An Architectural Challenge. In: Proc. of the Future of Software Engineering, FOSE, pp. 259–268 (2007)

[7] Muller, H.A., et al.: Autonomic Computing, Tech. Rep., Software Engineering Institute (April 2006)

[8] Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. Wiley (2000)

[9] Welsh, M., et al.: A Design Framework for Highly Concurrent Systems, T.R., UC Berkeley (2000)

[10] Welsh, M., Culler, D.: Adaptive Overload Control for Busy Internet Servers. In: Proc. of the 4th USENIX Conference on Internet Technologies and Systems, p. 4 (March 2003)

[11] Zhang, X., Lung, C.-H., Franks, G.: Towards Architecture-based Autonomic Software Performance Engineering. In: Proc. of the 4th Conference on Software Architectures, CAL (2010)

[12] Zhang, X., Lung, C.-H.: Improving Software Performance and Reliability with an Architecture-Based Self-Adaptive Framework. In: Proc. of IEEE COMPSAC (2011)