Recognising the Capacities of Dynamic Reconfiguration for the QoS Assurance of Running Systems in Concurrent and Parallel Environments

Wei Li
School of Information & Communication Technology
Central Queensland University
Rockhampton QLD 4702, Australia
Email: w.li@cqu.edu.au

Maolin Tang
Discipline of Networks and Communications
Queensland University of Technology
Brisbane QLD 4001, Australia
Email: m.tang@qut.edu.au

Abstract—**Recognizing the impact of reconfiguration on the QoS of running systems is especially necessary for choosing an appropriate approach to dealing with dynamic evolution of mission-critical or non-stop business systems. The rationale is that the impaired QoS caused by inappropriate use of dynamic approaches is unacceptable for such running systems. To predict in advance the impact, the challenge is two-fold. First, a unified benchmark is necessary to expose QoS problems of existing dynamic approaches. Second, an abstract representation is necessary to provide a basis for modeling and comparing the QoS of existing and new dynamic reconfiguration approaches. Our previous work [8] has successfully evaluated the QoS assurance capabilities of existing dynamic approaches and provided guidance of appropriate use of particular approaches. This paper reinvestigates our evaluations, extending them into concurrent and parallel environments by abstracting hardware and software conditions to design an evaluation context. We report the new evaluation results and conclude with updated impact analysis and guidance.**

Keywords-dynamic reconfiguration; QoS assurance; software maintenance; software evolution; quantitative analysis

I. INTRODUCTION

One important aspect of a running system is the Quality of Service (QoS) it is obliged to maintain to satisfy predefined domain requirements. In this paper, we use the end-user observable performance attributes: *throughput* and *response time* to measure the QoS of a running software system. Another important aspect is that, over its lifecycle, a system must change repeatedly to adapt to changing business needs and consequent changed requirements, differing from its original requirements. A *reconfiguration* refers specifically to a specific change at a specific point in the history of a system's evolution. Previous research [5], [11] shows that the combination of these two aspects, that is, maintaining the QoS of a system over its lifecycle, have never been addressed simultaneously. *Static reconfiguration* has significant impact on the QoS of a running system because it is performed out of band and involves stopping a running system, recompiling its binaries, rebuilding its data, altering its topology and then rebooting the system. *Dynamic reconfiguration* [5], involving upgrading/altering a system's functionality and topology at runtime via addition, deletion and replacement of components and connections, was initially proposed to increase service availability of evolving, running systems, taking a step towards addressing both aspects. However, Vandewoude et al. [11] indicated that *quiescence* [5], a characteristic of early approaches, has a significant impact on the QoS of a running system. That raised an issue: recognizing the impact of dynamic reconfiguration is essential to both aspects.

Recognizing the impact of various reconfiguration approaches has become a major concern [3] for dynamic evolution of running systems because for mission-critical services, any unpredictable change to QoS may be unacceptable; for 24/7 business services, the possibility of dissatisfaction increases due to the impaired QoS of the running systems. By *QoS assurance*, we refer to the capability of a reconfiguration approach to maintain some pre-determined minimum QoS level for a running system under reconfiguration (RSUR), even if there is some short term reduction of QoS. Such a notion leads to the possibility of enhancement of newer QoS assurance features.

The aim of this paper is to extend our previous work [8] of the evaluation of QoS assurance capabilities of existing dynamic approaches into concurrent and parallel environments and to explore the effectiveness of our key technologies of QoS assurance [7] in the face of concurrency and parallelism. Modern hardware and software conditions make parallel computing pervasive, even on personal computers. For example, the new intel® CORE™ i7 has 8 CPUs; the Java multi-threading provides concurrency for multi-tasking. Therefore, it is necessary to extend our previous research into concurrent and parallel environments, exploiting these enhanced hardware and software conditions for recognizing and minimizing the impact of dynamic reconfiguration on the QoS of RSURs.

The rest of this paper is structured as follows: Section 2 reviews related work and its limitations with respect to QoS assurance and impact evaluation. Section 3 reviews the theoretical aspects of QoS assurance and impact evaluation. In Section 4, we propose adaptive overhead control to adapt reconfiguration to varying CPU availability in terms of number of available CPUs and CPU usages. Section 5 details the design of a unified evaluation context. In section 6, we report the evaluation results and analysis. In section 7, we conclude by characterizing QoS assurance capacities of dynamic approaches in concurrent and parallel environments.

II. Related Work

Isolated evaluations of the impact of reconfiguration on the QoS of RSURs can be found in related work [1], [6], [10], [11]. These evaluations used local and isolated evaluation criteria, case studies and metrics. Due to the inability of representing a richer set of QoS problems, inconsistent criteria or metrics among these evaluations, it is difficult to gain a clear recognition of a wider range of dynamic approaches in terms of choosing a particular reconfiguration approach to satisfy a particular domain application.

Hillman & Warren [4] compared the service continuity (coexistence and concurrency) algorithm of Mitchell et al. [9] and the quiescence (blocking) algorithm of Warren [12] in terms of the wait-time that the components spent on waiting for data from its upstream updating components in the case study of a component-based router. The results showed that the former was better in maintaining the QoS of a RSUR. Truyen et al. [10] did a similar evaluation and concluded that coexistence and concurrency between the old and new sub-system had less disruption on the QoS of a RSUR. The significance of these studies was a start of quantitative comparison of dynamic approaches on their disruptions to the QoS of RSURs.

Some work tried to address a richer evaluation of reconfiguration approaches. Among them, Fung and Low [2] proposed an evaluation framework, in which the scope of impacts and performance characteristics were proposed as feature requirements for the evaluation of impacts that reconfiguration places upon the running systems. Survey of domain experts for the importance of the proposed features is the way of study, with a Wilcoxon signed-rank test to rank the importance of the features. The strength of the framework is a qualitative confirmation of the essential features for the evaluation of reconfiguration approaches. Among these features, dynamic change impact analysis ranked the highest, and therefore it confirmed the importance of recognizing change impact in order to appropriately use reconfiguration approaches in different situations.

III. The Theoretical Aspects of QoS Assurance and Impact Evaluation

Our previous work [7], [8] has advanced the research of dynamic reconfiguration from two aspects. First, a set of characteristics was defined to represent QoS assurance, to which corresponding technologies were proposed and practiced in provision of QoS assurance for RSURs. We confirmed that QoS assurance for dynamic reconfiguration was realizable under some acceptable constraints [7]. Second, exiting reconfiguration approaches were represented and realized onto a unified evaluation context by a set of abstract *reconfiguration strategies*, which were exposed to a rich set of QoS problems through a benchmark. A quantitative benchmarking of the exiting dynamic approaches demonstrated their capabilities in achieving QoS assurance for RSURs on a single-CPU environment [8].

A. Representation and Measurement of QoS

While the meaning of QoS is different in application domains, the QoS was represented by a set of performance characteristics: *global consistency*, *service availability*, *service continuity*, *stateless-equivalence* and *QoS-assurance* in our research of QoS assurance for RSURs. These characteristics enabled a qualitative classification of existing reconfiguration approaches [7]. Implementation of the QoS characteristics and integration through plug-ins enabled the representation of existing reconfiguration approaches onto a unified evaluation context [8].

On considering the direct QoS experience of end-users, the key evaluation metrics of QoS should be the throughput and response time of a running system. System *throughput* is the number of requests that the system can process for completion in a predetermined time interval; system *response time* is the time delay between a request submission and its completion as confirmed by its response.

B. Logical and Physical Conditions of QoS Assurance

Our previous research confirmed that coexistence of the old and new sub-system is a necessary condition to assure the continuity of workflow of a RSUR. By coexistence the new sub-system is brought into full effect before the shutdown and removal of the old sub-system, and therefore, the coexistence feature brings the benefit of logical continuity of workflow. The key technology to provide coexistence is Dynamic Version Management (DVM) [7], which assures the non-interference of workflow in a structure that is partially shared by the old and new sub-system during the coexistence period.

Our previous research also confirmed that the controllability of resource usage of reconfiguration is a necessary condition to physically assure the QoS of a RSUR. The capability of DVM is just a logical assurance of the QoS of a RSUR. Competition for resources confers reconfiguration the ability to physically decline the QoS of a RSUR. In a single CPU environment where the CPU is not saturated by the ongoing transactions, the *pre-emptive* scheduling was effective to minimize the impact of reconfiguration to zero. While the CPU is always saturated by the ongoing transactions, the *time-slice* scheduling was successfully applied in provision of controllability for the impact on the QoS of a RSUR according to given QoS requirements [8].

C. Representation of Reconfiguration Approaches

Benchmarking needs a representation of related work onto a unified platform. We have successfully designed and applied the abstract reconfiguration strategies [8] to represent the whole spectrum of the-state-of-the-art approaches to dynamic reconfiguration as summarized as follows.

- *avl-cpt*: applying quiescence or tranquillity to assure both application consistency and availability. Reconfiguration has the same capability as that of ongoing transactions to compete for resources.
- *con-cpt*: applying DVM to assure the logical continuity of the workflow of a RSUR. Reconfiguration has the same capability as that of ongoing transactions to compete for resources.
- *sel-cpt*: the same as that of *con-cpt* plus state-sharing to minimize the load of state transfer to zero and to acquire stateless-equivalence.
- *qos-pre*: the same as that of *sel-cpt* except that reconfiguration is restricted to use free CPU time only, i.e. replacing competitive scheduling with pre-emptive scheduling. This strategy is applicable to CPU non saturation only.
- *qos-ts*: the same as that of *sel-cpt* except that reconfiguration is restricted only runnable in a predefined time slot for each schedulable time-slice, i.e. replacing competitive scheduling with time-slice scheduling. This strategy is applicable to CPU saturation only.

D. Benchmark

Our proposed Data Encryption & Digital Signature System (DEDSS) was recognized by the research community as an effective benchmark to evaluate the QoS assurance capabilities of reconfiguration approaches in terms of: accommodation of stateful components; accommodation of both component dependency and independency; allowing concurrency and varying workload. The conceptual DEDSS was successfully implemented as a component-based prototype C-DEDSS (illustrated in Fig. 1) and a dataflow-based prototype F-DEDSS [8]. Both prototypes created the environment of stateful system needing state transfer; partially shared structure during the coexistence period; component dependency needing dependent updates. The benchmark is effective to expose a richer set of QoS problems that reconfiguration must deal with in order to maintain the QoS of a RSUR.

IV. QoS Assurance in Concurrent and Parallel Environments

QoS assurance in concurrent and parallel environments relies on both hardware and software conditions. The hardware condition in parallel environments enables the number of CPUs be a factor for QoS assurance. The argument is that the number of CPUs represents the parallel processing capacity of a system, and the reallocation of CPUs is applicable to the overhead control of reconfiguration.

Concurrency is a software condition to exploit parallel resources, and multi-threading is available from programming languages to support concurrency. In a single CPU environment, the capacity of reconfiguration for resource competition is always against ongoing transactions. This is because multi-threaded transactions supports concurrency but cannot support parallelism of multi-tasks on a single physical CPU. A successful control of the impact of overhead on the QoS of RSURs in single CPU environments is time restriction (pre-emptive scheduling and time-slice scheduling), which is effectively applied to CPU non-saturation or saturation situations respectively [7]. In multi-CPU environments, concurrency can be further exploited to service concurrent tasks in parallel, and therefore it enables a better condition to apply overhead control to reconfiguration.

In multi-CPU environments, QoS assurance can be two-tiers. The first tier exploits parallelism through optimized reallocation of CPUs between reconfiguration and ongoing transactions to minimize the impact from reconfiguration on the QoS of a RSUR. The second tier restricts reconfiguration for CPU usage to further control the competition capacity of reconfiguration and assures the QoS according to given requirements. The proposal of two-tier assurance aims at 'fine-grained' control of reconfiguration overhead by fully exploiting hardware and software conditions in concurrent and parallel environments.

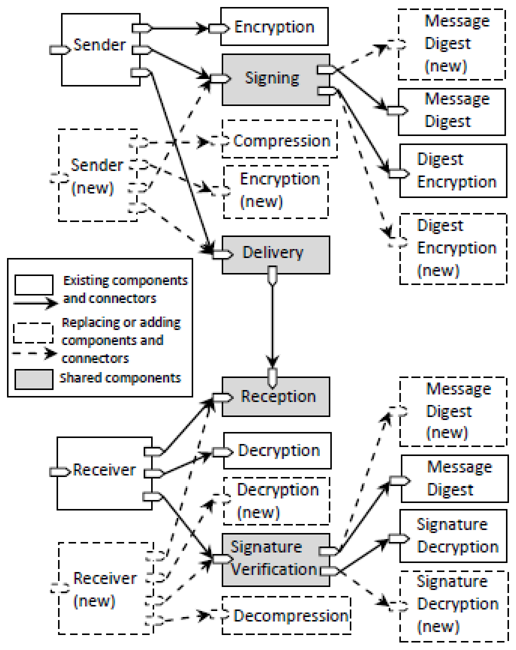

Figure 1. C-DEDSS [7]

A. *CPU Reallocation*

We suppose that the running priorities of all threads are same and therefore we suppose that each thread has the same competition capacity for resources. The reallocation of CPUs in a parallel environment is through threading. We also suppose that a parallel environment tries to reallocate CPUs in a balanced way:

- It allocates a free CPU to a newly created thread unless all CPUs are allocated. That is, a thread is using its own CPU and there is no competition for CPUs.
- While there are more threads than CPUs, the overall computing load is balanced for all CPUs. That is, a CPU can be reallocated to another thread from time to time.

The competition capacity of reconfiguration can be controlled by adjusting the number of reconfiguration threads according to the number of ongoing transaction threads. In an n-CPU environment, we suppose that the number of reconfiguration threads is r and the number of ongoing transaction threads is t. Consequently, we have the following conditions:

- If $r+t \leq n$, there is no competition of CPUs between reconfiguration and ongoing transactions.
- If $r+t>n$, reconfiguration competes CPUs with ongoing transactions. The competition capacity of reconfiguration against that of ongoing transactions is proportionate to r/t.

Based on the above conditions and given n and t, reconfiguration could have no competition if $t<n$ and $r \leq n-t$. Otherwise, if $t \geq n$, the minimum competition of reconfiguration will be achieved through $r=1$.

For example, in an 8-CPU environment, a reconfiguration can be 1-threaded, 2-threaded, or 4-threaded. If the ongoing transaction is already 4-threaded, the competition capacity of reconfiguration is therefore 0. However, if the ongoing transaction is already 8-threaded, the competition capacity of reconfiguration is 1/8, 2/8, and 4/8 against that of ongoing transactions. Therefore, the minimum competition of reconfiguration will be 1/8 with only one reconfiguration thread.

B. *CPU Time Reallocation*

The discussion of CPU time reallocation is under the condition that reconfiguration competition capacity has already been minimized by CPU reallocation i.e. $t \geq n$ and $r=1$. Under such a condition, to further minimize the competition capacity of reconfiguration, we investigate CPU usage and clarify it as saturation and non-saturation.

- While CPUs are non-saturated, they are idle from time to time. Therefore, we suppose that *pre-emptive* scheduling can minimise the competition capacity of reconfiguration. By prioritising the ongoing transactions, a reconfiguration is restricted to use free CPU time only and its competition capacity can be minimised to zero.
- While CPUs are saturated, we propose that the only way to control competition is to restrict reconfiguration for CPU time. We design *time-slice* scheduling to divide time into schedulable time-slices, of which each time-slice is further divided into two time-slots. In each time-slice a reconfiguration is put into runnable state in the *runnable time-slot* but into suspension state in the *sleeping time-slot*. By adjusting the length of the runnable time-slot, the competition capacity of reconfiguration is fully controlled and aligned to given QoS requirements.

V. THE DESIGN OF EVALUATION CONTEXT

The hardware and software platform that we use for the evaluation is an intel® CORE™ i7 8-CPU environment with Windows® 7 and Java SE 6. We suppose that the original system and the target system of the reconfiguration are on the same physical machine. To evaluate the impact of dynamic reconfiguration on the QoS of RSURs and cover various situations in terms of hardware (multi-CPU) and software (multi-threading) conditions and CPU usage (saturation and non-saturation), we have designed the following rules for the settings of evaluation context.

CPU non saturation: First, we set the evaluation context as $r+t \leq 8$, where r, $t \in \{1,2\ldots,7\}$ and make the 8 CPUs non-saturated. We suppose that CPU reallocation is effective to minimize the competition capacity of dynamic reconfiguration under such a condition. Consequently, the expected results will be no impact on the QoS of RSURs from all applicable reconfiguration strategies (checking *C* of Section 3) except for *avl-cpt*, which logically blocks the workflow and results in decline of the QoS of a RSUR at any situation. Second, we can enable t=8 but can set computing load to still make the 8 CPUs non-saturated. Such a condition necessitates the use of pre-emptive scheduling (*qos-pre*). The expected results will be no impact from *qos-pre* but with impact from all other applicable strategies.

CPU saturation: We set t=8 with the computing load to make the 8 CPUs saturated. Under such a condition, First, we set r=1 to minimize the competition capacity of reconfiguration by CPU reallocation. Second, CPU time reallocation is necessary to apply through time-slice scheduling. To demonstrate the controllability of time-slice scheduling, we set a fixed length of time-slices and use varying lengths of runnable time-slots. The expected results are smaller impact from a shorter runnable time-slot than those from a longer runnable time-slot scheduling.

Applying the above setting rules, we summarize the evaluation scenarios in Table 1, aiming at exposing the capacities of reconfiguration strategies on the QoS assurance of RSURs and covering various hardware and software conditions and CPU usages.

For the evaluation scenarios in Table 1, we use the benchmark of both C-DEDSS and F-DEDSS prototypes to expose the impact of dynamic reconfiguration on both system throughput and response time. The evaluation results and analysis are reported in next section.

VI. Evaluation Result and Analysis

To cover the evaluation scenarios in Table 1 with 2 prototypes (C-DEDSS and F-DEDSS) of the benchmark and 2 QoS metrics (throughput and response time) and 5 reconfiguration strategies (*avl-cpt*, *con-cpt*, *sel-cpt*, *qos-pre*, *qos-ts*), we conducted more than 100 evaluations on DynaQoS platform [7], [8]. We confirm that all evaluation results have the QoS impacts complying with the individual situations as expected in column 3 of Table 1, and the results are reproducible on both C-DEDSS and F-DEDSS, and there is positive correlation between throughput and response time, i.e. if the throughput of Strategy_1 is lower/higher than Strategy_2, the response time of Strategy_1 is correspondingly longer/shorter than Strategy_2. By page limit, we choose some typical evaluations of C-DEDSS to report the results.

TABLE I. The evaluation scenarios and expected impacts

CPU Usage	No. of reconfiguration threads verse No. of transaction threads	Expected QoS Impact
Non saturation	[r, t], where r+t≤8 and r, t∈{1, 2,…, 7}	Impact from *avl-cpt* only; no impact from *con-cpt*, *sel-cpt*
	[1, 8]	No impact from *qos-pre*; impact from *avl-cpt*, *con-cpt*, *sel-cpt*
Saturation	[1, 8]	Controlled impact from *qos-ts* uncontrolled impact from *avl-cpt*, *con-cpt*, *sel-cpt*

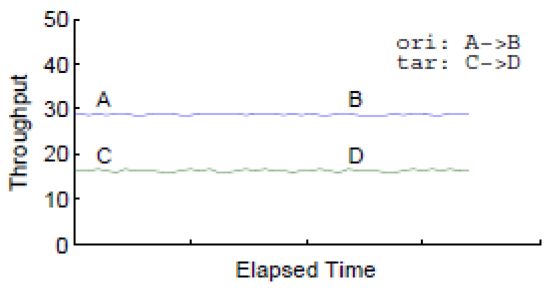

Figure 2. The base throughput of C-DEDSS (non saturation)

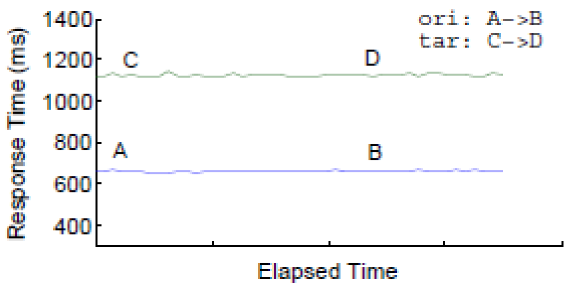

Figure 3. The base response time of C-DEDSS (non saturation)

A. The Base and Multiplied QoS Performance

Enabling only 1 transaction thread with non-saturation computing load, the base QoS performance of C-DEDSS is measured respectively by throughput (Fig. 2) and response time (Fig. 3) of the original system (*ori* lines) and the target system (*tar* lines) without any reconfiguration involved. In Fig. 2, the throughput of target system is lower than that of original system, or in Fig. 3, the response time of target system is longer than that of original system is due to the higher computing load of target system (checking *D* of Section 3 for the reconfiguration scenario).

Enabling *t* (*t*≤8) transaction threads, the *t*-times QoS performance is measured by throughput or response time of the original system and the target system without any reconfiguration involved. Under the same parameters as those in Fig. 2 and Fig. 3, the 4-times QoS performance of C-DEDSS is shown in Fig. 4 and Fig. 5 respectively for the running system's throughput and response time (*ori* and *tar* lines for the original and target system respectively). It is evident that the system's throughput (*ori* and *tar* lines in Fig. 4) is 4-times as large as those of the base throughput (*ori* and *tar* line in Fig. 2) but the system' response time (*ori* and *tar* lines in Fig. 5) remain unchanged (*ori* and *tar* lines in Fig. 3), reflecting no change of computing load of the application between Fig. 2 & Fig. 3, or Fig. 4 & Fig. 5.

B. Evaluation Results of CPU non Saturation

For [*r*, *t*] (*r*+*t*≤8) evaluation scenarios, we report the evaluation results of C-DEDSS of [1, 4] for throughput in Fig. 4 and response time in Fig 5. As expected, only *avl-cpt* has impacts on both throughput and response time. All other applicable reconfiguration strategies have no impact on either throughput or response time. We claim that for any [*r*, *t*] (*r*+*t*≤8), we obtained reproducible results as those of [1, 4]. These results confirmed the effectiveness of CPU reallocation in minimization of reconfiguration impact on the QoS of RSURs.

For the evaluation scenario of [1, 8], all 8 CPUs are used by the ongoing transactions. Under such a condition, all the previous strategies, i.e. *avl-cpt*, *con-cpt*, *sel-cpt* have impact on the 8-times QoS performance. The *qos-pre* is then applied to further exploit free CPU time for minimizing the impact of dynamic reconfiguration. The results are as expected, i.e. *qos-pre* has no impact but all other applicable strategies have impact on the QoS of the RSUR as reported in Fig. 6 and Fig. 7. These results confirmed the effectiveness of combining CPU reallocation and CPU time reallocation in minimization of reconfiguration impact on the QoS of RSURs.

C. Evaluation Results of CPU Saturation

For the evaluation scenario of [1, 8] and CPU saturation, all the previously applicable strategies, i.e. *avl-cpt*, *con-cpt*, *sel-cpt*, have impact on the 8-times QoS performance as reported in Fig. 8 and Fig. 9. Under such a situation, the QoS assurance strategy *qos-ts* is applied. To demonstrate the controllability of strategy *qos-ts*, we evaluate *qos-ts0.7* (a 700ms runnable time-slot in a 1000ms time-slice) and *qos-ts0.4* (a 400ms runnable time-slot in a 1000ms time-slice) scheduling. As expected, *qos-ts0.4* has smaller impact than *qos-ts0.7*. The impact ratios of *qos-ts0.7* and *qos-ts0.4* are proportionate to the lengths of their runnable timeslots as reported in Fig. 8 and Fig. 9. These results confirmed the effectiveness of time-slice scheduling for the controllability to reconfiguration impact on the QoS of RSURs.

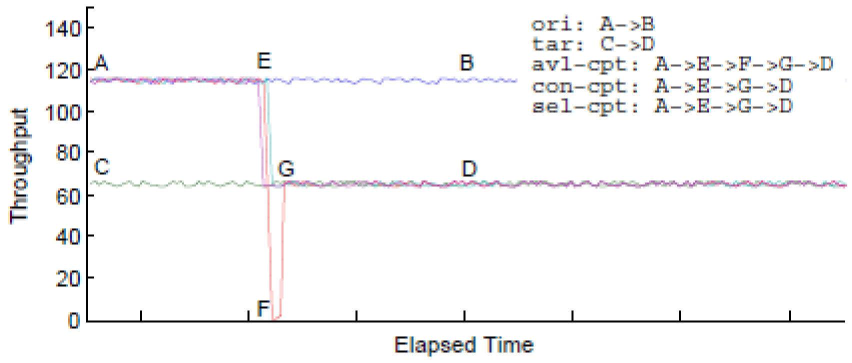

Figure 4. The C-DEDSS throughput of [1, 4] (non saturation) under the reconfiguration strategies

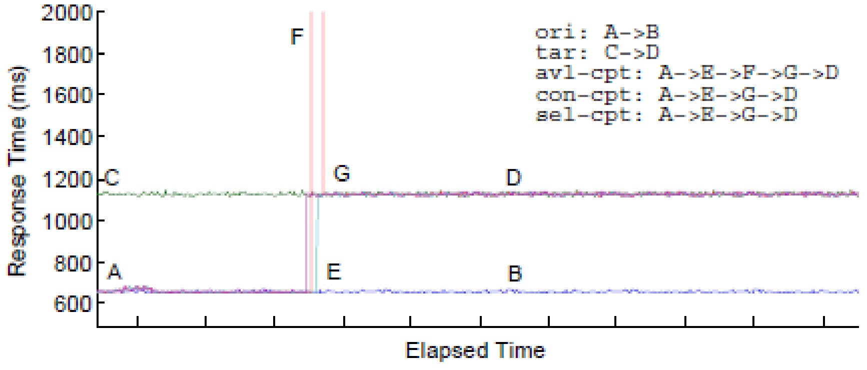

Figure 5. The C-DEDSS response time of [1, 4] (non saturation) under the reconfiguration strategies

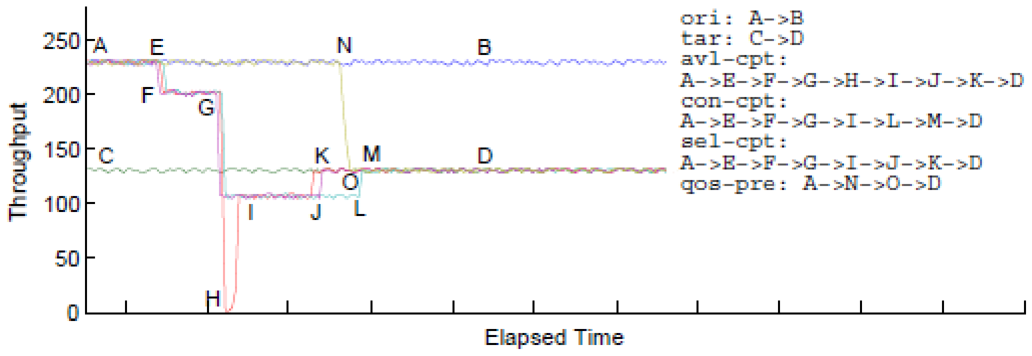

Figure 6. The C-DEDSS throughput of [1, 8] (non saturation) under the reconfiguration strategies

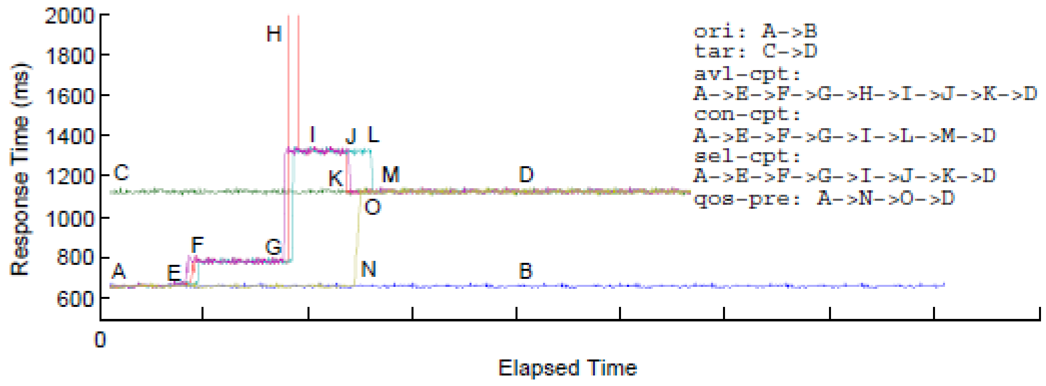

Figure 7. The C-DEDSS response time of [1, 8] (non saturation) under the reconfiguration strategies

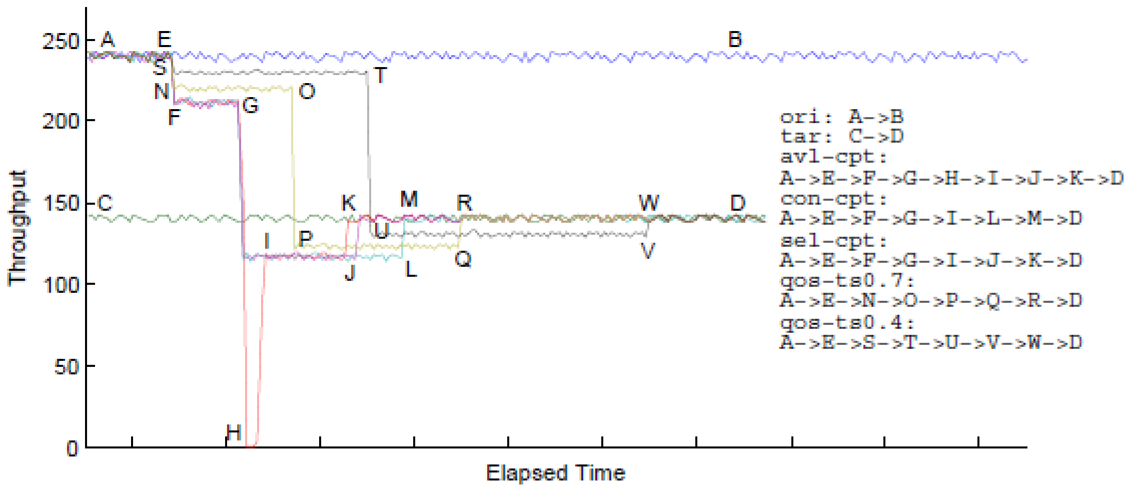

Figure 8. The C-DEDSS throughput of [1, 8] (saturation) under the reconfiguration strategies

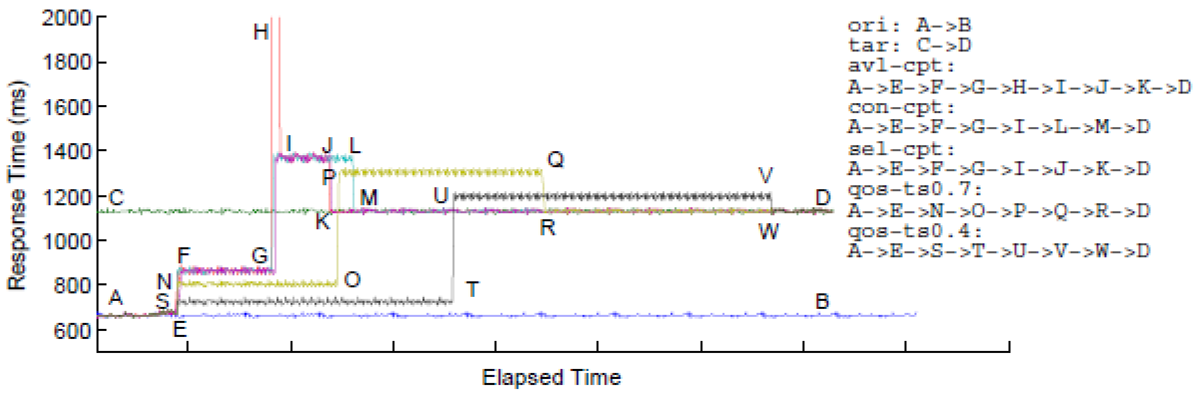

Figure 9. The C-DEDSS response time of [1, 8] (saturation) under the reconfiguration strategies

VII. Conclusion

Based on the reproducibility of the results on both C-DEDSS and F-DEDSS, and the positive correlation between throughput and response time for each reconfiguration strategy, we conclude evidently the impact of dynamic reconfiguration on the QoS of RSURs in concurrent and parallel environments. The necessary condition of QoS assurance is the logical continuity of workflow, i.e. the removal of blocking operation. To further exploit parallelism in multi-CPU environments, QoS assurance of dynamic reconfiguration can be fine-grained into two-tiers. First, while CPUs are not saturated: free CPUs or non free CPU but free CPU time, the impact of dynamic reconfiguration can be minimized to zero on RSURs via CPUs reallocation and/or CPU time reallocation (through pre-emptive scheduling). Second, while CPUs are always saturated, dynamic reconfiguration will definitely have impact on the QoS of RSURs. Time slice scheduling is the solution to gaining the controllability for the impact aligned with given QoS requirements. Our future work includes formal modeling of the theoretical framework for predicting the impact of and planning the QoS assurance for dynamic reconfiguration.

Acknowledgement

The authors would like to thank Dr. Ian Peake from RMIT University Australia for his proof-reading and other useful comments for the writing of this paper.

References

[1] S. Ajmani, B. Liskov, and L. Shrira, "Modular Software Upgrades for Distributed Systems," Proc. of 20th European Conf. on Object-Oriented Programming, Springer, pp. 452-476, 2006.

[2] K. Fung and G. Low, "Methodology Evaluation Framework for Dynamic Evolution in Composition-Based Distributed Applications," Journal of Systems and Software, Elsevier, 82(12), pp. 1950-1965, 2009.

[3] J. Hillman and I. Warren, "An Open Framework for Dynamic Reconfiguration," Proc. of 26th International Conf. on Software Engineering, IEEE Press, pp. 594-603, 2004.

[4] J. Hillman and I. Warren, "Quantitative Analysis of Dynamic Reconfiguration Algorithms," Proc. of International Conf. on Design, Analysis and Simulation of Distributed Systems, The Society for Modeling & Simulation International (SCS), 2004.

[5] J. Kramer and J. Magee, "The Evolving Philosophers Problem: Dynamic Change Management," IEEE Transactions on Software Engineering, IEEE Press, 16(11), pp. 1293-1306, 1990.

[6] M. Leger, T. Ledoux, and T. Coupaye, "Reliable Dynamic Reconfigurations in a Reflective Component Model," Lecture Notes in Computer Science, Springer, 6092, pp. 74-92, 2010.

[7] W. Li, "QoS Assurance for Dynamic Reconfiguration of Component Based Software Systems," IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2011.37, 2011.

[8] W. Li, "Evaluating the Impacts of Dynamic Reconfiguration on the QoS of Running Systems," Journal of Systems and Software, ELSEVIER, 84, pp. 2123– 2138, 2011.

[9] S. Mitchell, H. Naguib, G. Coulouris, and T. Kindberg, "A QoS Support Framework for Dynamically Reconfigurable Multimedia Applications," Proc. of International Conf. on Distributed Applications and Interoperable Systems, Kluwer, pp. 17-30, 1999.

[10] E. Truyen, N. Janssens, F. Sanen, and W. Joosen, "Support for Distributed Adaptations in Aspect-Oriented Middleware," Proc. of 7th International Conf. on Aspect-Oriented Software Development, ACM Press, pp. 120-131, 2008.

[11] Y. Vandewoude, P. Ebraert, Y. Berbers, and T. D'Hondt, "Tranquility: A Low Disruptive Alternative to Quiescence for Ensuring Safe Dynamic Updates," IEEE Transactions on Software Engineering, IEEE Press, 33(12), pp. 856-868, 2007.

[12] I. Warren, "A Model for Dynamic Configuration which Preserves Application Integrity," PhD thesis, Lancaster University, UK, 2000.