Heuristic Optimization of Scheduling and Allocation for Distributed Systems with Soft Deadlines

Tao Zheng, Murray Woodside

Dept. of Systems and Computer Engineering,
Carleton University, 1125 Colonel By Drive
Ottawa, Ontario, Canada K1S 5B6
{zhengtao, cmw} @ sce.carleton.ca

Abstract. This paper studies optimal deployment and priorities for a class of distributed real-time systems which have complex server tasks, many concurrent scenarios, operations with deterministic or stochastic execution demands, arbitrary precedence between operations, and hard or soft deadline requirements. The soft deadlines take the form of a required percentage of responses falling within the deadline. This work improves on an earlier optimization approach which was only applied to hard deadlines. As before, heuristic measures derived from solutions of layered queueing models are used to guide step-by-step improvement of the priorities and allocation, searching for a feasible solution which meets the soft deadline requirements. Effectiveness is demonstrated on a range of examples including thousands of individual cases.

1. Introduction

Performance specifications in many systems take the form of a soft deadline for each class *s* of responses, which execute a scenario also labeled *s*. A soft deadline is defined here as a requirement that the response time R_s for scenario *s* should satisfy a deadline D_s with some probability α_s. This may be written as:

$$P_s = \text{Prob}\ (R_s > D_s) \leq \alpha_s\ (\alpha_s \geq 0) \qquad (1)$$

A zero value for α_s defines a hard deadline; greater than zero a soft deadline.

This work describes a method for adjusting the priorities and allocations of the tasks in a distributed system, to satisfy such requirements. Unlike most work that simultaneously addresses priorities, allocation and deadlines, this work considers systems with some degree of randomness in their CPU demands, and with a *complex task structure* as described in [4]. The structure can include layers of servers to satisfy parts of the responses, with contention for service. Such systems and requirements are common in telecommunications systems, and in business systems. Examples include directory servers and proxy servers with layered structure, and e-commerce servers.

A great deal of work has been done on systems with hard deadlines (α_s = 0) and a flat task structure. Priority assignment and task (process) allocation were found as two important issues to meet deadline requirements for hard real-time systems deployed on multiprocessors. Unfortunately, it has been shown that the problem of assigning priority to end-to-end tasks [1], which have a chain of subtasks in a distributed system, and the problem of allocating tasks to processors [11] are both NP-hard problems. Efficient optimal solutions are not likely available, and heuristic algorithms must be created to find out feasible solutions. In previous related work, Tindel, Burns and Wellings used simulated annealing to find priority assignments and task allocations at the same time [19]. Garcia and Gonzalez Harbour proposed a Heuristic Optimized Priority Assignment (HOPA) algorithm which schedules transactions consisting of a chain of actions (subtasks) in a distributed system, to meet end-to-end hard deadlines [8]. Peng, Shin et al. proposed two branch-and-bound algorithms to allocate periodic tasks with precedence constraints to minimize the maximum response time [13]. Hou and Shin proposed an algorithm to find an assignment which maximizes the probability of meeting deadlines, and also to schedule the tasks, called the Module Allocation Algorithm [10].

To evaluate P_s for soft deadlines ($\alpha_s > 0$) and stochastic execution patterns, the distribution of the response time R_s must be found. Dingle, Harrison and Knottenbelt presented a technique for the numerical determination of response time densities in Generalized Stochastic Petri Net (GSPN) models [2]. Simulation methods can also be used, although they are more expensive in run-time.

El-Sayed et al in [4] considered the same problem, but for hard deadlines only. That paper described a heuristic optimization technique called the *Planner*, which used measures computed from a layered performance model to identify promising moves in the priority values and the allocations. The *Planner* met or surpassed other algorithms on test cases from the literature, and solved a large number of randomly generated problems of moderate size (with 16 tasks in four layers). The *Planner*, like this work, used simulation to determine P_s. Its application to soft deadlines was proposed in [4] but was not attempted.

The present work improves on the *Planner* in several ways, to be more effective on hard deadlines. The new version, called *Planner2*, is then evaluated on many small and large systems with hard and soft deadlines. On a large set of randomly generated stochastic systems, it was used to discover a property or characterization which indicates cases which are feasible for soft-deadline schedulability. The characterization takes account of the average processor utilizations, a latency factor, and the coefficient of variation of the execution demands of the tasks. The latency requirement increases with the variance of the demands.

This paper is organized as follows. Section 2 briefly introduces the Layered Queueing Networks (LQN) model which is used as the performance model in this paper. Section 3 provides the optimization algorithm of the *Planner2* based on the LQN simulation results and shows how it works. Section 4 applies the optimization approach to soft real-time systems with stochastic execution demands. Section 5 gives the conclusions.

2. The Layered Queueing Networks (LQN) Model

The layered queueing networks (LQN) model, presented by Woodside et al and others, is a performance model for systems with distributed software servers [5][6][17][20][22]. It extends queueing networks to model software servers and logical resources in a canonical way, including hardware devices, software processes, nested services, precedence constraints and multithreaded tasks.

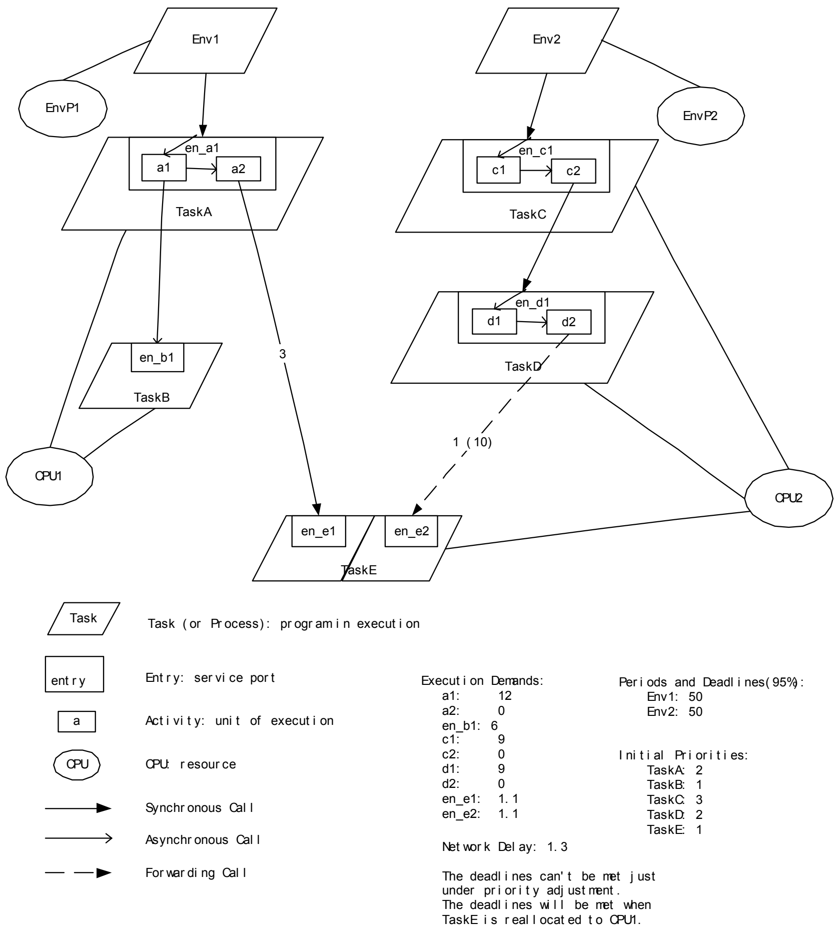

Fig. 1. An Example of an LQN model

In an LQN model, a *task* represents a software or hardware object which may execute concurrently, such as the tasks in a real-time system. A task has one or more *entries* which define its different services and are equivalent to classes in a queueing network. When a single-threaded task is busy serving a request to one entry, it cannot serve any other requests. An entry consists of some *activities* or *phases* which are the smallest execution units. Activities may have arbitrary precedence relationships (e.g. AND fork or join, OR fork or join) [7].

LQN tasks, entries, activities and interactions provide a description which is quite close to software architecture models. Figure 1 shows an example LQN model, with parallelograms representing tasks, rectangles on their interfaces for entries, internal rectangles for activities and their precedence, and arrows to denote interactions. Three types of interactions are described in LQNs: a synchronous call (shown in diagrams by a solid line with a filled arrowhead), an asynchronous call (a solid line with an open arrowhead), and a forwarding call (a dashed line with a filled arrowhead).

Both analytic and simulation solvers may be useful. In this paper, all the results are produced by an LQN simulation tool. The confidence interval for every result is no more than ±10%, meaning that all the results should be accurate within ±10% with 95% confidence. In most cases the actual results are much more accurate, of the order of 1%.

There are several ways to create LQN models. Petriu and Shen proposed a method to derive an LQN model from a Unified Modelling Language (UML) design model of the software, using the UML Profile for Schedulabilty, Performance and Time [16]. Petriu and Woodside described an algorithm to transform Use Case Maps (UCM) scenario models into LQN performance models [15]. El-Sayed generated LQN models and also guided the optimization from a model in a proprietary scenario language which is no longer available [3] (the lack of the inputs needed to use the *Planner* was one of the motivations behind this work).

Figure 1 is an example LQN model with two scenarios, both of which require 95% of the responses to meet the deadline. There are five tasks running on two CPUs, joined by a network (which is not shown). The activities and the network delays are stochastic, with exponential delays. *TaskE* is initially allocated on *CPU2*. The initial allocation and priorities are infeasible. Because there is higher communication cost between *TaskA* (on *CPU1*) and *TaskE* (3 calls per request) than between *TaskD* (on *CPU2*) and *TaskE* (1 call per request), the *Planner2* reallocates *TaskE* to *CPU1*, giving a feasible solution.

3. The Optimization Algorithm

The goal of *Planner2* can be stated as the minimization of a penalty function *V(A,P)* over a set *A* of task allocations and a set *P* of priorities for the tasks:

$$V(A,P) = \sum_{s} Criticality_s$$

$$\begin{array}{lll} Criticality_s = & 0 & P_s \le \alpha_s \\ Criticality_s = & e^{\beta(Ps-\alpha s)} & P_s > \alpha_s \end{array} \qquad (2)$$

V(A,P) is defined to be zero if the performance goal is met for all scenarios, or positive otherwise.

Briefly, *A* and *P* are initialized using heuristic algorithms, and the model of the system is solved. A set of scoring functions are computed to rank the scenarios, tasks and processors according to how they will contribute to improvement. These scores are used to select a change in priority or allocation. The search succeeds if $V(A,P) = 0$ or fails if $V(A,P) > 0$ and no step can be found that gives an improvement.

The original *Planner* is described in [3][4], and *Planner2* follows the same outline with changes in important details. For reasons of space, only the modified *Planner2* algorithm will be described, and the changes.

Algorithm

1. Initialize the configuration:
   a) Use the Multifit-Com algorithm [21] for the initial allocation,
   b) Give higher initial priority to tasks with fewer threads, on the same processor. Break ties by the Proportional-Deadline-Monotonic algorithm [18].
2. Check termination; use the *V(A,P)* metric defined by Eq. (2) to estimate the solution quality. If it is zero, go to 5. Otherwise go to 3.
3. Use the *TaskWaitingMetric* metric defined by Eq. (3) below to select a task with the largest value for priority increase. Estimate the solution quality. If *V(A,P)* is zero, go to 5. Otherwise go to 4.
4. Restore the configuration which has the smallest *V(A,P)* metric so far. Use the *ComGainMetric* metric defined in Eq. (4) below to select a task with the largest value, and a new CPU for it to move to. Estimate the solution quality. If *V(A,P)* is zero, go to 5. If failure conditions (on maximum total steps or on running out of alternatives to try) are satisfied, go to 6. Otherwise, go to 3.
5. Stop. A feasible solution is found.
6. Failed

The improvements over the *Planner* in [4] include:

- A different method for initializing the priorities based on the number of threads (higher priority to a task with fewer threads), and breaks ties by applying the Proportional-Deadline-Monotonic (PDM) algorithm. The original uses PDM entirely. The change was found to improve the success rate of the optimization.
- The *V(A,P)* metric in *Planner* used $e^{\beta P_s}$ if $P_s > \alpha_s$, giving a step at $P_s = \alpha_s$ whose height depends on α_s; here it is always a unit step. This normalizes the components of *V(A,P)* and balances the importance of violations in all scenarios.
- *Planner* used a single scoring function *TaskMetric* to identify the best task for changes, and this was the sum of two functions which are used separately in *Planner2*:

$$TaskWaitingMetric(t) = 1/U(t) \sum_s Criticality_s \; W(s,t) \qquad (3)$$

where *U(t)* is the utilization by task *t* of its processor, and *W(s,t)* is the total waiting time of task *t* in scenario *s* (these values are obtained from the LQN simulation report), and

$$ComGainMetric(t, c) = \sum_{\forall m \in nonLocalMsgs(c)} CommOV(m) - \sum_{\forall m \in LocalMsgs(t)} CommOV(m) \qquad (4)$$

where *nonLocalMsgs(c)* is the set of non-local physical messages of task *t* between CPU *c* and the local CPU of task *t*, *LocalMsgs(t)* is the set of local physical messages of task *t* and *CommOV(m)* is the overhead caused by the message *m*, plus the change in network delay.

In *Planner2* the first function is used for priority adjustment, and the second for task re-allocation.

- The priority levels of tasks are all forced to be distinct in *Planner2*, while equal priorities were allowed in [4]. Distinct priority levels decrease the sources of uncertainty in the response time.
- The priority adjustment strategy is simple in the *Planner*. The priority of the task with the worst *TaskMetric* metric will be raised one level higher. If the failure condition is met (i.e. the maximum step is reached), the priority adjustment stops. This strategy can hang up in a loop and fail to reach a better solution when one is available. To avoid this problem, the new priority adjustment strategy raises the priority level of the task with the largest *TaskWaitingMetric* by one level. If this priority combination occurred before (i.e., if a loop is found), then the priority level of this task will be raised to the highest priority level among all the tasks. If the new priority combination occurred twice before, then the priority adjustment stops and the *Planner2* considers task reallocation. This strategy is heuristic, but it worked well in the evaluations of section 4.1.

The table shown in Figure 2 follows the steps in the algorithm for the model shown in Figure 1. R_s, P_s and *V(A,P)* are given for the two scenarios.

4. Evaluation and Demonstration

Three sets of results will be described to demonstrate the success of the new version.

4.1 Evaluation of the New Priority Adjustment Strategies for Hard Deadlines

The new priority adjustment strategies were evolved partly to improve the optimization for hard deadlines. The improvement compared to [4] was evaluated by a battery of randomly generated cases with between two and eight independent periodic tasks on one processor. These test cases satisfy the requirements for rate monotonic scheduling [12], so the results could be compared with an exactly optimal solution. The fraction of the results found by *Planner* or *Planner2* over which are exactly optimal is called its “success ratio” in the results shown in Figure 3 below.

Step	Candidate Task	Actions And States	R_s	P_s	V(A,P)
0		CPU1: TaskA > TaskB CPU2: TaskC > TaskD > TaskE	33.781 23.405	0.092917 0.0444	1.903615
1	TaskE	Raise priority of TaskE CPU1: TaskA > TaskB CPU2: TaskC > TaskE > TaskD	27.953 24.165	0.0294 0.05095	1.014352
2	TaskD	The priority combination occurred before. Raise priority of TaskD to the highest level CPU1: TaskA > TaskB CPU2: TaskD > TaskC > TaskE	33.801 23.436	0.093402 0.04505	1.917514
3	TaskE	Raise priority of TaskE CPU1: TaskA > TaskB CPU2: TaskD> TaskE > TaskC	29.015 24.168	0.044317 0.051117	1.016896
4	TaskE	Raise priority of TaskE CPU1: TaskA > TaskB CPU2: TaskE > TaskD > TaskC	25.031 24.728	0.014767 0.054333	1.067153
5	TaskD TaskE	The priority combination occurred before. TaskD has the highest priority, its priority can't be raised any more. The best configuration is restored (step1). The TaskE has best benefit to be reallocated to CPU1 with the highest priority. CPU1: TaskE > TaskA > TaskB CPU2: TaskC > TaskD	18.138 25.225	0.0066833 0.049467	0

Fig. 2. Optimization steps of LQN model in Figure 1

Fourteen sets of cases were constructed with different combinations of the number of tasks and the total utilization (the utilization is the sum, over the tasks, of CPU time divided by the period). 50 cases were generated for each combination, giving 700 cases in total. To increase the difficulty of these cases, the initial priorities of the tasks were set to the reverse order to that assigned by the rate monotonic algorithm (thus, a task with a shorter period was given a lower priority). The results from the original *Planner* algorithm [4] and the new priority adjustment strategies are compared in Figure 3.

Number Of Tasks	Utilization	Success Ratio % (Original [4])	Success Ratio % (New)
2	0.82	100	100
2	0.90	100	100
3	0.77	100	100
3	0.90	100	100
4	0.75	98	100
4	0.90	91.67	100
5	0.74	94	100
5	0.90	80	100
6	0.73	98	100
6	0.90	40	100
7	0.72	96	100
7	0.90	42.86	100
8	0.72	94	100
8	0.90	33.33	100

Fig. 3. Test cases and results

The new strategies are a definite improvement. They have found a feasible solution in every case which is feasible under rate monotonic scheduling, whereas the original algorithm had a significant number of failures, especially with larger task numbers and utilizations.

4.2 Evaluation of *Planner2* with Soft Deadlines

The *Planner2* was equally successful with random CPU demands and soft deadlines, and on more complex systems with layered servers. Two evaluations are described here, first for a large set of randomly generated layered systems, and second for an application case study with a realistic architecture.

4.2.1 Layered Randomly-generated Cases with Stochastic Demands

Figure 4 shows an LQN introduced in [4] to demonstrate the robustness of optimization on hard real-time applications with deterministic CPU demands that were selected randomly. The evaluation is extended here to soft deadlines and stochastic execution demands.

The main characteristics of the randomly generated parameters are:

- Every scenario has a fixed period and deadline,

  $$Deadline_s = Demand_s * L \qquad (6)$$

  where L is the *laxity factor*, taking values between 1.9 and 6

- The deadline requirement is that the deadline miss rate is no more than 5%
- The average utilization of all the processors is adjusted to take a selected fixed value for a given case, chosen between 0.4 and 0.8
- The coefficient of variation *CV* of the execution demand was fixed, taking values 0.0 (deterministic), 0.1, 0.5 or 1.0 (exponential) for all the tasks.

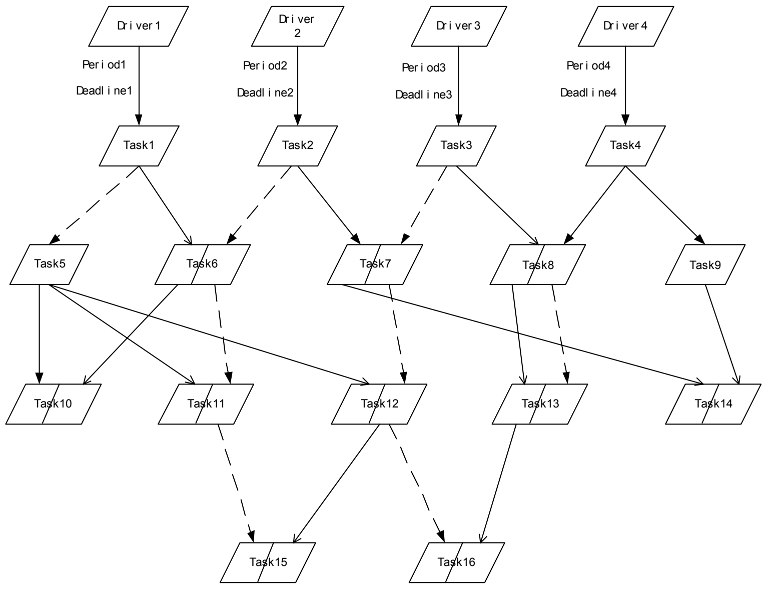

Fig. 4. Random statistical models

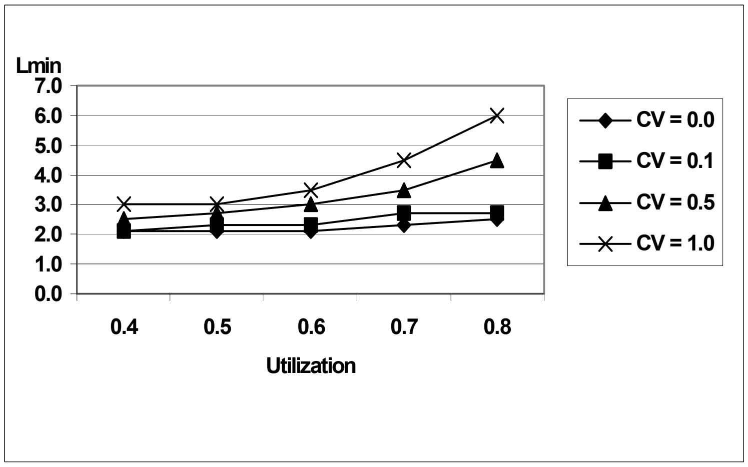

Fig. 5. Minimum laxity factor value providing feasibility for different combinations of the coefficient of variation and the processor utilization

There were altogether 240 different combinations for the coefficient of variation, laxity factor and utilization. 50 cases were generated for each combination, and all 12000 cases were optimized by the *Planner2*. Figure 5 shows some of the results.

For each combination of utilization and *CV*, Figure 5 shows the minimum laxity factor value (L_{min}) for which a feasible allocation and priority solution was found for all 50 cases. We observe that:

- For a given utilization, the required minimum laxity factor increases as the coefficient of variation increases, and as the utilization increases.
- The minimum laxity factor with large coefficient of variation (e.g. 1.0) increases much faster than that with small coefficient of variation (e.g. 0.0) as the utilization increases. This indicates the extreme difficulties to meet deadline requirements with large coefficient of variation and high utilization.

  For larger laxity values, the fraction of cases that is found to be infeasible is positive and increases with laxity value.

The Figure can be interpreted as a heuristic guideline for feasibility of a set of soft deadlines based on a system's utilization and coefficient of variation values. For example, with an average utilization of 0.6 and $CV = 1.0$, the laxity factor for all tasks should be at least 3.5. However this guideline is only for a 95% success rate and provides no guarantees.

4.2.2 RADS Bookstore Model

This example is a simplified e-commerce site described in [14] by Petriu and Woodside, called the RADS Bookstore model. The model describes a 3-tier client-server system (client, application and database tiers) with stochastic behaviour. The customer has 7 scenarios: browsing the stock, viewing a detailed item description, adding or removing items to or from shopping cart, checking out the items in shopping cart, registering and logging into the RADS bookstore. The administrator can update the inventory and fill the outstanding back orders. Figure 6 is the simplified LQN model of RADS bookstore, originated from a diagram in [14]. The model has been adapted as follows:

- The set of Customers represented by the Customer task have a random think time between requests to the system, and a probability for making each type of request. There is one Administrator task.
- Each scenario for the Customer and Administrator is governed by a pseudo task in the second layer, running on a pseudo processor *ScenarioProc*. The pseudo tasks are used to collect response times and to set deadlines.
- The scenario deadlines for Customer are all set to 500ms, and the scenario deadlines for Administrator are set to 6000 ms.
- The deadline miss rates for scenarios are required to be no more than 10%.

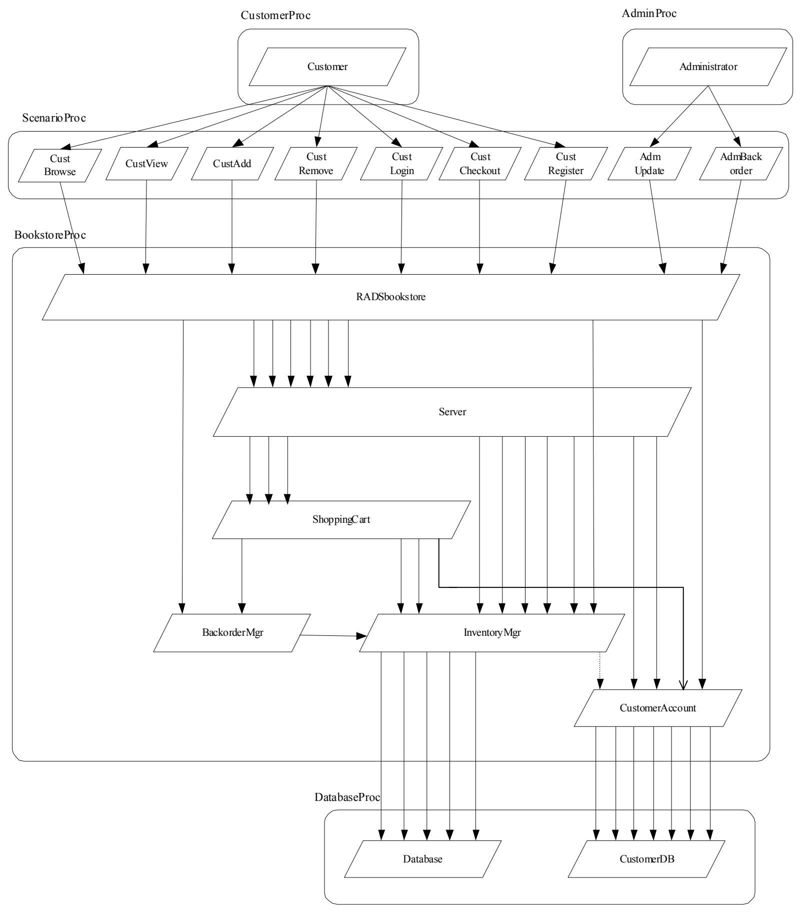

Fig. 6. Simplified LQN model of RADS Bookstore

The model was analyzed with 50, 100, 150, 200, 250, 300 and 350 customers. Because there is only one processor in the application tier, there is no task reallocation, and priority adjustment is the only optimization option. In the baseline model all the tasks on processor *BookstoreProc* and processor *DatabaseProc* are scheduled by the FIFO discipline (i.e. all the tasks are assigned the same priority).

The optimized model will be compared to the baseline model. It turns out that the Customer scenarios easily meet their deadlines, so the experimental results in Figure 7 only show the miss rates for the two Administrator scenarios. These are greatly improved by the optimization. In the baseline model, the miss rates for the two administrator's scenarios increase rapidly when the number of customers increases, and the deadline requirements couldn't be met when the number of customers is 200 or more. In the optimized model, the miss rates of the two Administrator scenarios are held roughly constant, and the deadline requirements are met for all cases.

This case study shows how the optimization approach can be usefully applied to find a runtime configuration for a complex hierarchical client-server system with soft deadlines and stochastic behaviour. The performance of the result is comparable to the redesign proposed in [15], which was determined with considerable analysis and required restructuring the database subsystem. This is an outstanding success for an automated procedure.

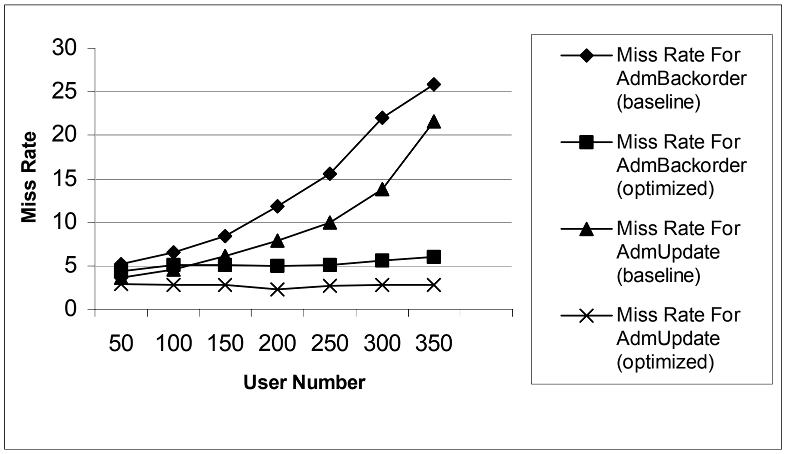

Fig. 7. Miss rates of baseline and optimized model for scenario AdmBackorder and AdmUpdate

5.0 Conclusions

An improved method for optimizing the configuration of a layered real-time system has been described. It is intended to be useful to software designers who wish to evaluate a software design “at its best”, without the effort of manually tuning the deployment. It adjusts the priorities of tasks competing for a processor, and the allocation of tasks to processors, searching for a feasible configuration (meaning, one that meets soft deadlines on percentiles of responses). It can equally be used to configure systems with hard deadlines, to be met by 100% of responses. The percentiles can be different for different scenarios.

The *Planner2* is significantly better than its predecessor at finding feasible configurations for hard deadlines. It successfully configured thousands of cases with soft deadlines as well, with complex task structures. It successfully configured a realistic task system for e-commerce, without intervention.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Council of Canada through its program of Discovery Grants, and by the Ontario government through its OGSST program of scholarships.

References

[1] R. Bettati, “End-to-end scheduling to meet deadlines in distributed systems”, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, March 1994.

[2] N. J. Dingle, P. G. Harrison, W. J. Knottenbelt, “Response time densities in Generalized Stochastic Petri net models”, In Proceeding of the Third Workshop on Software and Performance, Rome, July, 2002

[3] H.M. El-Sayed, D. Cameron, and C.M. Woodside, "Automated performance modeling from scenarios and SDL designs of distributed systems", In Proc. of the Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), Kyoto, April 1998

[4] H.M. El-Sayed, D. Cameron, C. M. Woodside, “Automation Support for Software Performance Engineering”, Proc. Joint Int. Conf. on Measurement and Modeling of Computer Systems (Sigmetrics 2001/Performance 2001), Cambridge, MA, 2001, ACM order no. 488010, pp 301-311.

[5] G. Franks, A. Hubbard, S. Majumdar, D. Petriu, J. Rolia, and C.M. Woodside, “A toolset for performance engineering and software design of client-server systems”, Performance Evaluation, 24 (1-2):117-135, November 1995.

[6] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, “Performance analysis of distributed server systems”, Proceedings of The Sixth International Conference on Software Quality, Ottawa, Canada, October 28-30, 1996, pp. 15-26.

[7] G. Franks, “Performance analysis of distributed server systems”, Ph.D. thesis, Dept. of Systems and Comp. Eng., Carleton University, Dec. 1999.

[8] J.J.G. Garcia and M. Gonzalez Harbour, “Optimized priority assignment for task and messages in distributed hard real-time systems”, Proc. IEEE Workshop on Parallel and Distributed Real-Time Systems, California, pp. 124-132, April 1995.

[9] Mark K. Gardner, “Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems”, Thesis of Doctor of Philosophy, in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 1999

[10] C.J. Hou and K.G. Shin, “Allocation of periodic task modules with precedence and dead-line constraints in distributed real-time systems”, in Proc. of the Real Time system symposium, pp. 146-155, 1992

[11] J.Y.T. Leung, J. Whitehead, “On the complexity of fixed-priority scheduling of periodic real-time tasks”, Performance Evaluation, 2, (4), pp. 237-250, Dec. 1982.

[12] C.L. Liu, J.W. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment”, J. Assoc. Computing. Mach., v 20, pp 46-61, 1973.

[13] D.T. Peng and K.G. Shin, “Static allocation of periodic tasks with precedence constraints in distributed real-time systems”, In Proc. of the 9th Intl. Conf. On Distributed computing systems, pp. 190-198, 1989.

[14] Dorin Petriu, Murray Woodside, "Analysing Software Requirements Specifications for Performance", . Third Int. Workshop on Software and Performance, Rome, July 2002

[15] Dorin C. Petriu, Murray Woodside, "Software Performance Model from System Scenarios in Use Case Maps" , International Conferences on Modelling Techniques and Tools for Computer Performance Evaluation, p141-p158, 2002

[16] Dorina C. Petriu, Hui Shen "Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML specifications, International Conferences on Modelling Techniques and Tools for Computer Performance Evaluation, p159-p177, 2002

[17] J. R. Rolia and Kenneth Sevcik, "The method of layers", IEEE Transactions on Software Engineering, Vol. 21, No. 8, pp. 689-700, 1995.

[18] Jun Sun, "Fixed Periodic Scheduling of Periodic Tasks with End-To-End Deadlines", Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, March 1996.

[19] K.W. Tindel, A. Burns, and A.J. Wellings, "Allocating hard real-time tasks: an NP hard problem made easy", Real-Time Systems, 4(2):145-165, June 1992.

[20] C.M. Woodside, "Throughput calculation for basic stochastic rendezvous networks", Performance Evaluation, Vol. 9, No. 2, pp. 143-160, 1989.

[21] C.M. Woodside and G.M. Monforton, "Fast allocation of processes in distributed and parallel systems", IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 2, Feb. 1993.

[22] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, "The stochastic rendezvous network model for performance of synchronous client server-like distributed software", IEEE Transactions on Computers, 44(1):20-34, January 1995.