Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

TOWARDS ADAPTIVE SCHEDULING OF TASKS IN TRANSACTIONAL WORKFLOWS

Manolis Marazakis
Christos Nikolaou

Department of Computer Science, University of Crete and
Institute of Computer Science, FORTH
P.O. Box 1385, GR 71110 Heraklion, GREECE

ABSTRACT

This paper discusses dynamic workload management in transaction processing systems where the workload consists of multiple classes of units of work, including workflows comprised of interdependent tasks. Business requirements specify that differing levels of service must be provided to different classes of work, thus it is natural to specify performance goals per work class, that reflect the business requirements for the work class as well as the inherent resource demands of the units of work. Adaptive algorithms have been proposed for the satisfaction of performance goals of transaction classes. Scheduling the execution of complete workflows, which are multi-transaction units of work, is complicated by the need for task coordination, due to both control and data flow dependencies among tasks. Current transaction processing monitors provide infrastructure for the coordination of tasks by means of queueing facilities. We draw on previous work on goal-oriented resource management to design adaptive task scheduling algorithms. A detailed simulator of transaction processing systems with a queueing facility has been developed, with the specific aim to study the performance for workloads that include multi-transaction units of work.

1 INTRODUCTION

This paper discusses dynamic workload management in transaction processing systems where the workload consists of multiple classes of units of work, including workflows comprised of interdependent tasks. It is necessary to satisfy concurrency atomicity, failure atomicity, and permanence guarantees for complex business activities. These properties can be guaranteed by executing activities as transactional units of work, which guarantee the *ACID* properties (Breitbart et al. 1993). Scheduling the execution of complete workflows which are multi-transaction units of work is complicated by the need for task coordination, due to both control and data flow dependencies among tasks. Business requirements specify that differing levels of service must be provided to different classes of work, thus it is natural to specify performance goals per work class, that reflect the business requirements for the work class as well as the inherent resource demands of the units of work.

Transaction processing (TP) monitors (Gray 1978, Gray and Reuter 1993, Bernstein 1990) provide infrastructure for the development of data-intensive applications. They provide support for running tasks, for handling persistent communication among tasks, for communication with users, and for access to multiple database systems. Current-generation TP monitors (Kageyama 1989, Speer and Storm 1991, Sherman 1993) provide only basic primitives that can be used for workflow management, however next-generation TP monitors (Dayal et al. 1993) will be required to include facilities for modeling and managing the execution of complex business activities. Such activities (Dayal et al. 1990, Dayal et al. 1993) typically consist of many steps, are of long duration, may require access to multiple, possibly heterogeneous, shared databases, and may involve interaction with multiple individuals in an enterprise. A business activity may start with a step, which in turn may trigger asynchronous and deferred steps (Dayal et al. 1990). Each step may invoke applications that execute transactions over one or more databases. Support for extended transaction models (Elmagarmid 1992, Biliris et al. 1994) will be required to capture application-specific semantics.

It is therefore important to study how the currently available infrastructure can be used to support workflows, and how to manage classes of workflows from a performance point of view. We focus on "system-oriented" workflows, according to the classification in Georgakopoulos et al. (1995), as they constitute the

basis on which workflow management services have to be built.

We draw on previous work (Ferguson et al. 1993) on goal-oriented transaction scheduling and routing to design adaptive task and queue management algorithms. A detailed simulator of multiple processor transaction processing systems with a queueing facility has been developed, with the specific aim to study the performance of algorithms for workloads that include multi-transaction units of work. The focus of this paper is on task scheduling policies, however we also outline our work-in-progress in the area of routing policies that take into consideration queue placement.

The rest of the paper is organized as follows: Section 2 presents our model of execution for transactional workflows, and introduces the concept of performance goals for workflow classes. Section 3 discusses alternative scheduling policies that attempt to satisfy performance goals over classes of units of work, while Section 4 discusses issues arising in a multi-system environment where routing of individual steps of units of work has to be performed. Section 5 briefly describes the simulator that we designed and implemented in order to evaluate the performance impact of transaction scheduling and routing policies when transactions are steps of multi-transaction units of work, and presents a preliminary evaluation of the scheduling policies of Section 3. Section 6 concludes the paper, outlining directions for further research.

2 MODEL OF WORKFLOW EXECUTION

Units of work of class WC_p, $p = 1, \ldots, P$, are assumed to consist of a sequence of component transactions ("steps") $T_{p1}, \ldots, T_{pn_p}$. Each such step T_{pt} is classified, by a workload characterization module, as an instance of some transaction class C_i, $i = 1, \ldots, K$. For each class of multi-transaction units of work, there is a *performance goal* which is stated as the requirement that the average response time of instances of that class does not exceed a certain limit G_p. This notion is based on the concept of "service-level agreements" (Noonan 1989), and describes a requirement for explicit "quality-of-service" guarantees. This type of performance goals is statistical in nature, as they do not require that specific deadlines are met by individual units of work, which is the case with real-time systems (Abbot and Garcia-Molina 1988).

The transaction processing system is assumed to incorporate N nodes, $S_1, \ldots, S_N$. There are front-end nodes where user requests for the execution of multi-transaction units of work arrive, coupled in a Shared-Nothing architecture. Each transaction in the sequence of steps that makes up a multi-transaction unit of work is routed to a node for service.

For each transaction class, a workload characterization module provides a profile of average resource consumption. This profile characterizes the expected behavior of transactions in each transaction class in the N-node transaction processing system, and includes the average CPU work generated on system S_k by a class C_i transaction routed to system S_l, the average number of times a class C_i transaction routed to system S_l "visits" the CPU at system S_k, the expected I/O delay of a class C_i transaction, the expected communication delay of a class C_i transaction routed to system S_l, and the expected I/O delay due to writing log records at all sites for a class C_i transaction routed to system S_l. This profile can be used to compute an estimate of the service time (ignoring queueing delays) of a transaction of class C_i, which is a measure of the intrinsic cost of processing a transaction of class C_i.

The work in Ferguson et al. (1993) assumes that each transaction class has been statically assigned an average response time goal, which reflects business requirements for the transaction class while taking into account resource demands. Since in this paper we are interested in multi-transaction units of work, we consider a dynamic assignment of performance goals for transactions that appear as steps in workflows. Specifically, we use the service time estimate $RT_{service}(T_{pt})$ for steps T_{pt} to allocate "portions" $g(T_{pt})$ of the predefined goal G_p for class WC_p workflows to individual steps, as follows:

$$g(T_{pt}) = G_p \cdot \frac{RT_{service}(T_{pt})}{\sum_{t=1}^{n_p} RT_{service}(T_{pt})},$$

where T_{pt}, $t = 1, \ldots, n_p$ are the steps of a class WC_p workflow. Assigning such subgoals to steps allows us to set up feedback mechanisms in our effort to satisfy performance goals for workflow classes.

The execution model for transactions that need to access data on multiple nodes is similar to that supported in the IBM's CICS TP monitor (Kageyama 1989). A "primary" transaction is initiated at a node selected by the front-end to execute the application program that issues database calls, and "secondary" transactions are started on other nodes to process the requests forwarded ("function-shipped") to them by the primary transaction.

Although this is a restricted model of workflow, it often arises in practice when a long-duration activity is executed as a chain of several transactions, rather than as a single long-duration transaction (Garcia-Molina and Salem 1987). Long-duration transactions can seriously affect overall system performance (Gray

1981). In a distributed environment, chaining of transactions may be required if not all the nodes that process the request are available at the same time. Moreover, from our performance point of view, this class of workflows is particularly important as it frequently appears as a building block for more complex workflows.

It is assumed, as in Bernstein et al. (1990), that a sequence of "server tasks" executes the sequence of transactions for the completion of the request, and each task registers with a *request* and a *reply* queue, managed by the queue manager of the underlying TP monitor. The application cannot rely on local variables and (non-persistent) data structures to record the state of the request across transaction boundaries, due to the possibility of failures. Therefore, when information, such as a request for transaction initiation, or data items produced by a transaction, is to cross transaction boundaries, a server task must store it in the request queue for the recipient. Queue managers store records submitted by transactions as uninterpreted byte strings in volatile or persistent storage, and allow retrieval in FIFO or priority order, or even by content. In the event of a failure, which causes a transaction to be aborted, the initiation request for that transaction is returned to its (persistent) input queue, thus the sequence of transactions that service the original request is not broken.

A disadvantage of this model is that the execution of requests is not serializable (although the execution of component steps is serializable). However, for many important applications (Garcia-Molina and Salem 1987, Gray 1981) it is possible to relax strict serializability requirements. If this is not acceptable, then some mechanism is required for controlling interference between concurrently executing workflows. This paper ignores this issue, focusing on performance management aspects of the model.

Each transaction initiation request is kept in a queue until the transaction monitor at the service node can allocate a task to service the request. The time that the transaction initiation request for T_{pt} spends waiting until it is bound to a task directly affects the response time of T_{pt}, and therefore the overall response time for the whole WC_p request. For this reason, a priority index is assigned to each transaction initiation request. The service node allocates tasks to service transaction requests in increasing priority index order.

3 TASK SCHEDULING POLICIES

We focus on preemptive, priority-driven CPU scheduling policies, and consider their performance potential in environments where the objective is to fullfil service-level agreements, expressed as the requirement that average response time per workload class does not exceed a class-specific upper bound. We consider a number of alternative schemes for the assignment of priorities to workflow steps (assuming that the CPU scheduler selects tasks for service in *increasing* priority index order, and does round-robin scheduling for tasks with the same priority index). The priority assignment for each transactional step in general has to consider the goal specification for the workflow class, the current status of the class regarding goal satisfaction, and some form of feedback on the response time of previous steps, together with the inherent resource requirements of the step. The alternative policies that we consider differ in the choice of factors to consider for scheduling decisions.

RR: Round-robin scheduling of transactions (all transactions have equal priority). We expect this policy to provide an upper bound for the evaluation of more elaborate scheduling policies, as it does not consider any knowledge about the workflow or the individual transactions.

Static: The priority of a transactional step T_{pt} of a class WC_p workflow is set to the goal G_p for the class. This policy requires no knowledge of the resource requirements of steps, taking into account only the goal for the workflow. This static assignement favors units of work with short response time goals, similarly to the earliest-deadline-first policy for real-time tasks (Abbot and Garcia-Molina 1988).

PI: The priority of a transactional step of class WC_p is set to $\frac{1}{PI_p}$, where PI_p is the current value of the performance index for the class. The *performance index* PI_p of class WC_p is defined as

$$PI_p = \frac{R_p}{G_p},$$

where R_p is the current estimate of average response time for units of work of class WC_p. R_p is updated upon termination of a unit of work T of this class as follows:

$$R_p \leftarrow (1-\alpha) \cdot R_p + \alpha \cdot R(T),$$

where $R(T)$ is the response time of T and α is a constant ($0 \leq \alpha \leq 1$) that weighs the relative importance of recent measurements of response time against measurements of response time for units of work that completed farther back in time. *PI* is therefore an adaptive policy, as it considers satisfaction of class performance goals, favoring transactions of classes that at this particular decision instant are more probable to exceed their specified response time

goal. This policy was proposed in Ferguson et al. (1993) for single-transaction units of work.

Weighted-PI (W-PI): The priority of a step t of workflow class WC_p is set to $\frac{w_t}{PI_p}$, where w_t is a weight coefficient computed as in the pseudo-code shown in Figure 1, and PI_p is the current value of the performance index for WC_p. The coefficient w_t allows for priority adjustment as the workflow proceeds with the execution of its component steps, and each step completion provides more insight as to whether the workflow will meet its performance goal. The performance index is included in the priority calculation so as to incorporate some knowledge about the average response time of class WC_p. End-users are primarily interested in the response time of workflows rather than individual steps, therefore it is necessary to consider the current status of workflow classes regarding their performance goals as well as the response time of previously completed transactional steps, so as to maintain some control over CPU contention among concurrent workflows. The w_t coefficients depend on the response time of previously completed steps of a workflow, as well as on the inherent resource requirements of remaining steps.

Weighted-Task-Goal (W-T): This is a simplification of *W-PI*. The priority of a step t of class WC_p is set to $w_t \cdot g(t)$. This policy is expected to be less responsive to workload dynamics than *W-PI*, as it does not explicitly consider the current status of workflow classes regarding goal satisfaction.

Let us denote $prio_{pt}$ the priority assigned to transactional step T_{pt}. The pseudo-code in Figure 1 defines how the w_t coefficients are adjusted after each transactional step completion in the workflow, based on feedback from previous steps.

The function $F(w_m, r_{pt}, g(T_{pt}))$ referenced in Figure 1 determines the amount by which the w_m weight coefficients are adjusted when a transactional step misses its response time goal:

$$F(w_m, r_{pt}, g(T_{pt})) \;:=\; w_m \cdot \frac{r_{pt} - g(T_{pt})}{g(T_{pt})}.$$

The intuition behind this definition of function $F(w_m, r_{pt}, g(T_{pt}))$ is that when a step in a workflow misses its goal, thus making the performance goal for the workflow class harder to achieve, the next step should be "expedited" so as to cover for the previous step's delay.

4 ROUTING POLICIES

In general, minimizing average response time will not satisfy performance goals. Rather, the objective of workload management should be to maximize the system throughput at which class goals are still met.

```
t := 1 ;
w_m := g(T_pm) / Σ_{m=1}^{n_p} g(T_pm), for m = 1, ..., n_p ;
while (t ≤ n_p) {
    Initiate-Transaction(T_pt, service-node, prio_pt);
    /* Assign priority index prio_pt to the transaction
     initiation request for T_pt. Use prio_pt as the priority
     for task T_pt, relay request to service-node */
    Wait-for-Transaction-Completion(T_pt) ;
    /* Wait for service-node to send notification of the
     completion of T_pt. Service-node also sends the
     actual response time, r_pt, of T_pt. */
    w_m := g(T_pm) / Σ_{m=t+1}^{n_p} g(T_pm), for m = t + 1, ..., n_p ;
    if(r_pt > g(T_pt)) {
        /* T_pt missed its response time goal */
        w_{t+1} := w_{t+1} − Σ_{m=t+2}^{n_p} F(w_m, r_pt, g(T_pt)) ;
        w_m := w_m + F(w_m, r_pt, g(T_pt)),
            for m = t + 2, ..., n_p ;
    }
    t := t + 1 ;
}
```

Figure 1: Priority Assignment Based on Feedback

In Ferguson et al. (1993) a set of goal-oriented routing algorithms for multiple systems coupled in a Shared-Nothing architecture are presented. For each incoming transaction, the effect of assigning it to each available node is evaluated by computing an estimate of the response time of the transaction (if it is routed to that node), and using it to compute the effect of this routing decision to the goal satisfaction of all transaction classes.

These algorithms can be extended to take advantage of information about how a workflow is doing with respect to its goal that becomes available as successive steps complete. We are currently working on modifications of the response time estimate used by these algorithms so as to take into consideration the fact that a transaction may have to access intermediate results produced by a previous transaction in the workflow that has been placed in a recoverable queue at some node, so as to enable the routing algorithms to take into account the affinity to queued data that a workflow begins to exhibit as its steps complete their execution one after the other. This type of affinity is quite different from affinity to database partitions that is determined primarily by the design of the database, as affinity to queued data evolves with time. We plan to evaluate by simulation combi-

nations of these modified routing algorithms with the scheduling policies discussed in Section 3.

An important issue is queue placement, i.e. the selection of the node at which the records produced by a workflow step are to be stored so that a subsequent transactional step can access them. As a first approximation, we are considering a static assignment of queues to nodes. This seems to be a natural choice, especially for business environments where workflows have to access several database servers, as in the case of workflows that require access to data from different departments of an enterprise.

5 EXPERIMENTAL EVALUATION

In this section we present a number of simulation experiments for the evaluation of the scheduling algorithms of Section 3, for a workload consisting of two workload classes $WC1$, $WC2$. For this evaluation, we consider a single-node system.

5.1 Simulation Model

Our evaluation is based on a detailed simulation model of a transaction processing facility that supports a queueing facility. The simulator is built in C on top of a threads-based simulation support library, *PARASOL* (Neilson 1991), and a parser that processes a high-level description of the system configuration and workload in order to configure the simulated run-time environment according to user specifications.

The simulator fully emulates concurrency control (two-phase locking with deadlock detection), buffer management (global LRU), logging (including group commit), CPU scheduling, I/O scheduling, queue management, and distributed two-phase commit. It implements the transaction and workflow execution model described in previous sections, and collects performance-related statistics per workload (transaction and workflow) class. For multi-transaction units of work, we provide an explicit specification of control and data flow using a C-function that implements this flow as a series of transactions submitted in a chain fashion.

The main metrics that we consider in our evaluation of scheduling policies for multi-transaction units of work are the maximum performance index over all workflow classes, and the *violation distance* metric, which is a convenient single-figure performance characterization defined as

$$VD := \sum_{WC_p} max\{\frac{R_p - G_p}{G_p}, 0\},$$

where R_p is the average response time, *as measured during system operation*, for workflow class WC_p that has a response time goal G_p.

We model a system with a 50 MIPS CPU, and a scheduling quantum of 10 msec. I/O access delay is modeled in detail by the simulator, taking into account device-specific cost parameters (like the average seek delay). In our experiments, the average I/O access delay was approximately 20 msec. The database pages are allocated to 2 disks, in a round-robin manner, and there is a seperate log disk. We consider a skewed access pattern for the database: 80% of the accesses refer to 20% of the data pages. The size of the database is taken to be 50,000 pages, while the database buffer can hold up to 20% of the database pages, and uses the LRU replacement policy. We assume that 80% of all accesses are updates. We are interested in studying the performance impact of the scheduling policies of Section 3 under high load conditions.

There are two workflow classes, with equal arrival frequency, and 4 different transaction classes. Workflow class $WC1$ consists of a chain of 3 transactions, of classes A, B, D respectively, whereas workflow class $WC2$ consists of a chain of 5 transactions, of classes C, A, B, D, C respectively. We model transactions as a sequence of database accesses, interleaved with bursts of CPU processing. Table 1 defines the (synthetic) transaction classes A, B, C, D by giving their average application pathlength, and the minimum and maximum number of database accesses that they perform. Table 2 shows the cost for several system functions that affect the response time of transactions, measured as instruction counts (pathlengths).

Table 1: Transaction Class Profiles

transaction class	avg application pathlength	min/max DB calls
TX-CLASS-A	40,000	1 - 4
TX-CLASS-B	60,000	1 - 8
TX-CLASS-C	80,000	1 - 12
TX-CLASS-D	100,000	1 - 16

5.2 Simulation Results

Several simulation experiments were carried out in order to investigate the impact of the scheduling policies presented in Section 3, for a number of users that simultaneously submit multi-transaction units of work for service. We assume a closed queueing model, i.e.

Table 2: Pathlengths for System Functions

system function	# instructions
task initiation/termination	15,100
data manager interface	2,000
DB call processing	4,000
data I/O processing	10,000
logging I/O processing	5,000
commit	12,000

each user is modeled as a source that submits a request, waits until the system services the request, and then waits for a period of time ("think time") before submitting the next request. Think time is assumed to be exponentially distributed, with a mean of 5 seconds, and the number of users is set to 50. The objective is to get a "snapshot" of the simulated system's performance under high load conditions. For each reported data point we performed 5 simulation runs for an interval of 3600 seconds of simulated time. We set the response time goal G_{WC2} for class $WC2$ to 5 seconds, and vary the goal G_{WC1} for class $WC1$. Table 3 summarizes the measurements of the performance indices for $WC1$, $WC2$, for the simulated system configuration. Figure 2 presents the measurements of

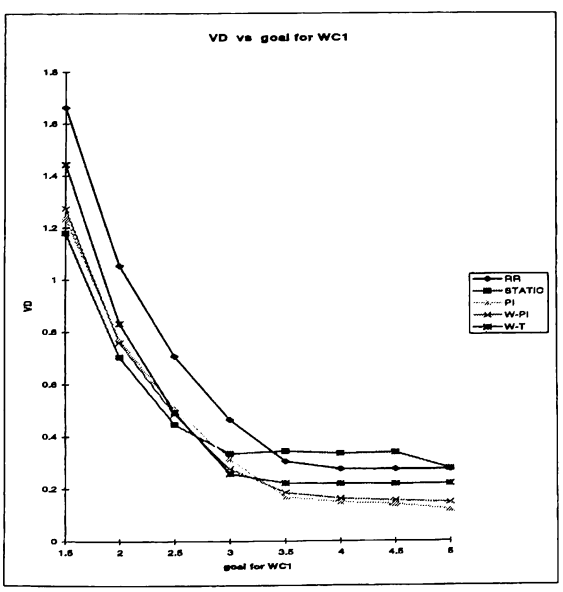

Figure 2: Violation Distance for Workflow Classes

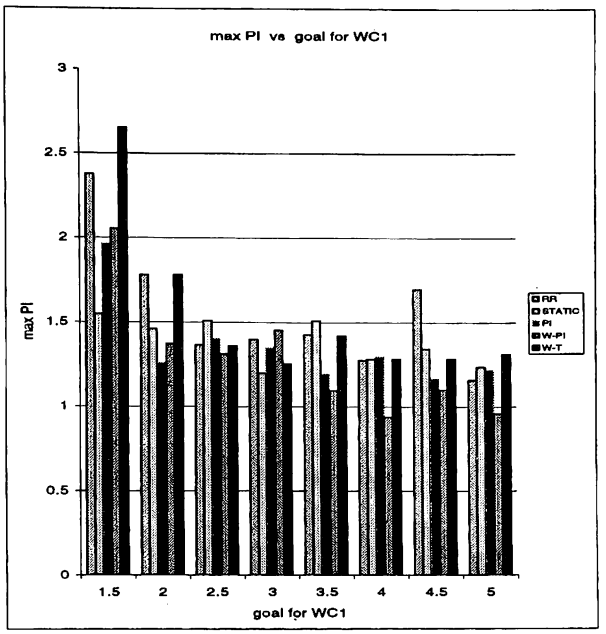

Figure 3: Maximum Performance Index

the violation distance metric. Figure 3 summarizes the measurements of the *maximum* performance index, thus allowing us to visualize the impact of the proposed scheduling policies on performance goal satisfaction.

Depending on system load, the system configuration, and the resource allocation policies adopted, a performance goal specification may not be achievable. The scheduling policy has to balance the allocation of CPU cycles to transactions executing on behalf of the competing workflow classes.

The *W-PI* policy is the only one that achieves to keep class average response times within 10% of the specified performance goals, when the goal for $WC1$ is set to 3.5 sec, or higher. *W-PI* considers the goal specification for the workflow class, the current status of the class regarding goal satisfaction, and some form of feedback on the response time of previous steps, which makes it more responsive to workload dynamics than the other policies that we studied. The *RR* policy provides a baseline for comparison, while the rest of the policies consider only subsets of the factors that *W-PI* considers, therefore they cannot fully support goal-oriented scheduling. *W-T* does not consider goal satisfaction of workload classes, re-

lying mainly on feedback about each individual step, given the subgoal assigned to each step. For unachievable goal specifications, this policy fails, as subgoals are computed as portions of the specified class goal. This failure is aggravated by the fact that at high load the simulated system exhibits a high lock conflict rate. *CLASS-PI* appears to be quite stable, as it tracks the performance index and adapts accordingly. *W-PI*, which also tracks the response time of individual steps, achieves better performance for achievable goal specifications ; however, for "unrealistic" goal specifications its dependence on feedback about subgoals makes it lose its edge over *CLASS-PI*. On the other hand, *STATIC* achieves better performance than the adaptive policies for short response time goals for $WC1$, which cannot be satisfied by the system. The problem with *STATIC* is that it is biased against classes with higher response time goals. When the goal for $WC1$ is set to 3 seconds, or higher, Table 3 shows that *STATIC* keeps the performance index for $WC1$ below 1.0, while $WC2$ misses its goal.

Table 3: Measured Class Performance Indices

G_{WC1}	Performance Indices					
	Class	*RR*	*STATIC*	*PI*	*W-PI*	*W-T*
1.5	WC1	2.378	1.548	1.963	2.053	2.652
	WC2	1.136	1.241	1.046	1.243	1.166
2.0	WC1	1.781	1.459	1.216	1.372	1.779
	WC2	1.113	1.319	1.256	1.358	1.149
2.5	WC1	1.363	1.430	1.403	1.311	1.359
	WC2	1.311	1.509	1.133	1.210	1.293
3.0	WC1	1.191	0.745	1.032	1.454	1.065
	WC2	1.397	1.197	1.346	1.366	1.255
3.5	WC1	1.097	0.945	1.131	0.843	0.828
	WC2	1.426	1.509	1.194	1.095	1.420
4.0	WC1	0.884	0.755	0.808	0.868	0.802
	WC2	1.274	1.283	1.296	0.939	1.283
4.5	WC1	0.931	0.599	0.970	0.768	0.570
	WC2	1.697	1.344	1.165	1.100	1.283
5.0	WC1	0.754	0.566	0.649	0.586	0.610
	WC2	1.157	1.236	1.216	0.957	1.311

6 CONCLUDING REMARKS AND DIRECTIONS FOR FUTURE WORK

It is our view that goal-oriented resource management policies can offer considerable benefits in workflow processing environments. Our work on extensions of techniques previously studied for single-transaction units of work to a restricted but nevertheless important class of workflow models provides motivation for further work in this area. It is clear however that a more formal treatment of the scheduling problem for workflows is required. To this end, an important result is presented in Bhattacharya et al. (1991), which presents a stochastic optimization formulation for the scheduling problem for units of work that have to visit a number of queueing nodes, assuming a single server that has to be allocated to one of the nodes at a time, in a non-preemptive manner. An instance of this abstract system model could be a single CPU and a number of software servers, such as database managers. A disadvantage is that the solution technique is very complex, making practical implementation a challenge. Furthermore, it is not clear how to extend this result to the case of multiple resources, as in a distributed transaction/workflow processing system.

As pointed out in Denning (1994), the process of estimating performance metrics for complex units of work is complicated by the need to correlate the individual steps of a unit of work. The design and implementation of low overhead, scalable monitoring schemes is therefore particularly important, as an essential aid to adaptive workload management. For distributed systems, state propagation algorithms, as well as sampling techniques to control monitoring overhead, must be studied. Another important problem is the interaction between various adaptive resource managers. As a specific example, we are studying the combined use of an adaptive task scheduler with an adaptive transaction router, in an environment where queues managed by the TP monitor provide inter-transaction communication. As different resource managers have control over different control parameters, which may adversely affect one another, it is important to study stability issues. Interacting resource managers are essential for integrated performance management within a framework that enables the specification of performance goals per workload class.

ACKNOWLEDGMENTS

The work reported in this paper was funded by the LYDIA (ESPRIT III BRA 8144) research project.

REFERENCES

Abbot, R., and H. Garcia-Molina. 1988. Scheduling real-time transactions: a performance evaluation. In *Proceedings of the 14th VLDB Conference*, pp. 1-12.

Bernstein, P. 1990. Transaction processing monitors. *Communications of the ACM*, vol. 33, no. 11, pp. 75-86.

Bernstein, P., M. Hsu, and B. Mann. 1990. Implementing recoverable requests using queues. In *Proceedings of the 1990 ACM SIGMOD Conference on Management of Data*, pp. 112-122.

Bhattacharya, P., L. Georgiadis, and P. Tsoucas. 1991. Optimal adaptive scheduling in multi-class $M/GI/1$ queues with feedback. In *Proceedings of the 29th Allerton Conference on Communication, Control and Computing*, pp. 312-321.

Biliris, A., S. Dar, N. Gehani, H. V. Jagadish, and K. Ramamritham. 1994. ASSET: a system for supporting extended transactions. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 44-54.

Breitbart, Y., A. Deacon, H. J. Schek, A. Sheth, and G. Weikum. 1993. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *ACM SIGMOD Record*, vol. 22, no. 3, pp. 23-30.

Dayal, U., M. Hsu, and R. Ladin. 1990. Organizing long-running activities with triggers and transactions. In *Proceedings of the 1990 ACM SIGMOD Conference on Management of Data*, pp. 204-214.

Dayal, U., H. Garcia-Molina, M. Hsu, B. Kao, and M. C. Shan. 1993. Third generation TP monitors: a database challenge. In *Proceedings of the 1993 ACM SIGMOD Conference on Management of Data*, pp. 393-397.

Denning, P. 1994. The fifteenth level. In *Proceedings of the 1993 ACM SIGMETRICS Conference*, pp. 1-4.

Elmagarmid, A.K. (editor). 1992. *Database transaction models for advanced applications.* San Fransisco: Morgan Kaufmann.

Ferguson, D., C. Nikolaou, L. Georgiadis, and K. Davies. 1993. Satisfying response time goals in transaction processing systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pp. 138-147.

Garcia-Molina, H., K. Salem. 1987. Sagas. In *Proceedings of the 1987 ACM SIGMOD Conference on Management of Data*, pp. 245-259.

Georgakopoulos, D., M. Hornick, and A. Sheth. 1995. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119-153.

Gray, J. 1978. Notes on database operating systems. In *Operating Systems: An Advanced Course*, ed. R.Bayer, R.M. Graham and G. Seegmuller. Springer-Verlag Lecture Notes in Computer Science, vol. 60, pp. 393-481. New York: Springer-Verlag.

Gray, J. 1981. The transaction concept: virtues and limitations. In *Proceedings of 7th International VLDB Conference*, pp. 144-154.

Gray, J., and A. Reuter. 1993. *Transaction processing: concepts and techniques.* San Fransisco: Morgan Kaufmann.

Kageyama, Y. 1989. *CICS Handbook.* New York: Intertext Publications/McGraw-Hill.

Neilson, J. E. 1991. PARASOL: a simulator for distributed and/or parallel systems. Technical Report SCS-TR-192, Carleton University, Canada.

Noonan, J. 1989. Automated service level management and its supporting technologies. *Mainframe Journal*, October 1989, pp. 102-103.

Sherman, M. 1993. Architecture of the Encina distributed transaction processing family. In *Proceedings of the 1993 ACM SIGMOD Conference on Management of Data*, pp. 460-463.

Speer, T. and M. Storm. 1991. Digital's TP monitors. *Digital Technical Journal*, vol. 3, no. 1, pp. 18-32.

AUTHOR BIOGRAPHIES

MANOLIS MARAZAKIS is a graduate student at the Computer Science Department of the University of Crete, Greece and a research assistant at ICS-FORTH, Heraklion, Crete. His research interests include transaction management, and workload management of parallel and distributed systems.

CHRISTOS NIKOLAOU is an Associate Professor at the Computer Science Department of the University of Crete, Greece and a Researcher at ICS-FORTH, Heraklion, Crete, where he is the head of the Parallel and Distributed Systems Group. Coordinator of LYDIA, ESPRIT III BRA 8144 on load balancing for distributed systems. From 1981 until 1992 he was a Research Staff member and then a Research Staff Manager at the IBM T.J. Watson Research Center in Yorktown Heights New York. He was awarded the IBM Outstanding Innovation Award for his scientific contributions on goal-oriented workload management. He holds an M.Sc. and a Ph.D. from Harvard University. He is an IEEE Senior Member and Chairman of the Executive Committee of ERCIM (European Consortium on Informatics and Mathematics). His research interests include parallel and distributed systems and databases, and economic models for resource allocation.