Scheduling Workflows on a Cluster of Memory Managed Multicore Machines

Hira K. Shrestha[†], Nicolas Grounds[‡], Jason Madden[‡], Matthew Martin[‡], John K. Antonio[†], Jay Sachs[‡], Josh Zuech[‡], and Carlos Sanchez[‡]
[†]School of Computer Science, University of Oklahoma, Norman, OK, USA
[‡]RiskMetrics Group, 201 David L. Boren Blvd, Suite 300, Norman, OK, USA

Abstract— *Workflows are modeled with directed acyclic graphs in which vertices represent computational tasks, referred to as requests, and edges represent precedent constraints among requests. Associated with each workflow is a deadline that defines the time by which all computations of a workflow should be complete. Workflows are submitted by numerous clients to a centralized scheduler that assigns workflow requests to a cluster of memory managed multicore machines for execution. The objective of the scheduler is to minimize missed workflow deadlines. The characteristics of workflows are assumed to vary along several dimensions, including: arrival rate, periodicity, degree of parallelism, and number of requests. Five scheduling policies are evaluated; four of these policies are known from the literature and one policy is newly proposed. The advantages and disadvantages of each policy is determined through simulation studies.*

Keywords: Workflow Scheduling; Performance Modeling; Simulation; Automatic Memory Management

1. Introduction

An application supported by a service-oriented architecture (SOA) is modeled in this paper as a workflow graph (WFG), which is a directed and acyclic graph that defines precedence constraints among service requests required by the application. WFGs can vary greatly in size and structure. For example, a small WFG may contain just a few requests (i.e., vertices) while a large WFG may contain thousands of requests. Regarding structure, at one extreme a WFG may represent a single chain of requests in which no two requests may be executed in parallel. At another extreme, the structure of a WFG may contain numerous independent chains of requests in which requests belonging to distinct chains may be executed in parallel.

In the framework considered here, WFGs are assumed to be submitted by multiple sources (i.e., clients) to a central scheduler. Associated with each submitted WFG is a deadline that defines the time by which all requests of the WFG should complete execution. The main objective of the scheduler is to assign requests of submitted WFGs to machines of the cluster so as to reduce missed deadlines of all WFGs.

The remainder of the paper is organized in the following manner. Section 2 includes an overview of related work. Section 3 describes the simulation environment developed to evaluate different scheduling policies. Section 4 describes specific scheduling policies considered in this paper. Section 5 provides the results of simulation studies, followed by concluding remarks in the final section.

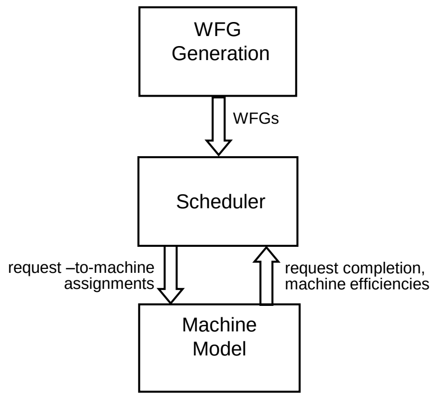
2. Background and Related Work

Previous related work is reviewed in three broad areas: (1) machine modeling and simulation environments; (2) automatic memory management; and (3) scheduling and load balancing.

Considerable work has been published related to modeling of machines in distributed environments. Much of the past research in this area has focused on modeling and predicting CPU performance, e.g., [1], [2]. The machine model described in the present paper (refer to Section 3.4) relies on assumed knowledge of the characteristics of the requests (i.e., computational tasks); it is similar in a sense to the SPAP approach proposed in [1].

In memory managed systems, the effect of long and/or frequent garbage collections can lead to undesirable – and difficult to predict – degradations in system performance. Garbage collection tuning, and predicting the impact of garbage collections on system performance, are important and growing areas of research, e.g., [3], [4], [5], [6], [7]. To estimate the overhead associated with various garbage collectors, experiments were designed and conducted in [4], [5] to compare the performance associated with executing an application assuming automatic memory management versus explicit memory management. The machine model proposed here accounts for the overhead associated with automatic memory management.

Formulations of realistic scheduling problems are typically found to be NP-complete, hence heuristic scheduling policies are generally employed to provide acceptable scheduling solutions, e.g., refer to [7], [8], [9], [10], [11], [12]. The scheduling evaluations conducted in the present paper account for the impact that garbage collection has on a machine's performance. Examples of other memory-aware scheduling approaches are described in [7], [13].



WFG Generation
WFGs
Scheduler
request –to-machine assignments
request completion, machine efficiencies
Machine Model

Fig. 1: Major components of the simulation environment.

Load balancing involves techniques for allocating workload to available machine(s) in a distributed system as a means of improving overall system performance. Examples of both centralized and distributed load balancing approaches are described in [8]. The scheduling framework developed in the present paper incorporates aspects of load balancing in the sense that the scheduler only assigns requests to machines that have loading factors (for CPU and memory) that are below defined thresholds.

3. Simulation Environment

3.1 Overview

The block diagram of Fig. 1 illustrates the three major components of the simulation environment. Each of these components is described in detail in Subsections 3.2 through 3.4.

3.2 WFG Generation

A WFG is a directed acyclic graph that is composed of parallel and sequential combinations of request chains (RCs). An example WFG is shown on the left side of Fig. 2. The vertices of the graph represent requests and the directed arcs denote precedence constraints that exist between requests. This WFG contains five RCs.

A WFG is a hierarchical structure that can be defined in a recursive manner by introducing the concept of a compound node. A compound node represents parallel instances of a common RC. The parallel RCs associated with a compound node represent instances of the same chain of requests that are to be executed with different input data sets. The right side of Fig. 2 represents the WFG using three compound nodes. The parallel RCs of a compound node could themselves be sequences of compound nodes, thus enabling the representation of WFGs of greater depth than the example shown in the figure.

The primary function of the component labeled WFG Generation in Fig. 1 is to provide synthetically generated

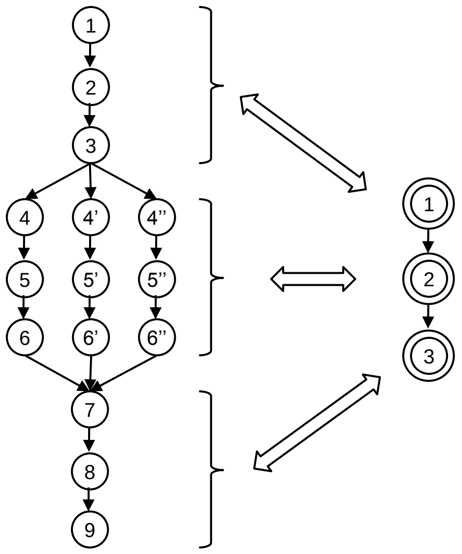


Fig. 2: Sample WFG (left) and its representation as a sequence of compound nodes (right).

Table 1: Definitions of CPU and heap memory requirements for request r.

$C_r > 0$	C_r is the total number of CPU cycles required to execute r on the fastest unloaded machine.
$D_r \geq C_r$	D_r is the ideal execution time duration of r on the fastest unloaded machine.
$U_r = C_r/D_r$	U_r is the CPU utilization factor of r.
$M_r > 0$	M_r is the maximum reachable heap memory requirement of r.
$A_r \geq M_r$	A_r is the total heap space allocated by r.
$G_r = A_r/M_r$	G_r is the garbage generation factor of r.

WFGs to the Scheduler for the purpose of evaluating scheduling policies. The WFG generation process used in this paper is probabilistic. Parameters for WFG generation rates, WFG structure, and CPU and memory requirements of requests that compose a WFG are defined for each WFG type generated. Table 1 summarizes the notation and definitions of basic computational and memory requirements for request r.

The CPU utilization factor of r, $U_r = C_r/D_r$, can be no greater than unity and no less than zero. A request having a CPU utilization factor of unity is typically referred to as a CPU-bound request, e.g., refer to [1].

The garbage generation factor of r, $G_r = A_r/M_r$, is a relative measure of how much garbage is generated by request r. The smallest possible value of G_r is unity, which corresponds to the extreme case in which a request does not generate garbage. Large values of G_r correspond to requests that generate garbage at a relatively high rate. This parameter is defined as the "total allocation to maximum reachable ratio" in [4], [5].

3.3 Scheduler

The Scheduler component of Fig. 1 takes WFGs as input and assigns requests of the WFGs to machines of the cluster for execution. In making assignment decisions, the Scheduler can make use of computational and memory requirements assumed to be known and available for each request. Having access to such information is realistic in the assumed dedicated environment in which off-line profiling and/or historical logging can be performed to collect/estimate these data. Associated with each WFG is a single timing deadline, and the Scheduler can also make use of WFG deadline requirements in making request scheduling decisions.

When a WFG arrives at the scheduler, it is initially placed in a pool that holds all WFGs that have not yet completed execution. The compound nodes of each WFG are considered in order and are expanded by the Scheduler to expose one or more parallel RCs. At any point during the execution of a WFG, the requests associated with one or more RCs are considered by the Scheduler for assignment to machines. The Scheduler tracks the status of each request (associated with RCs currently under consideration) according to one of the following state values: "completed," "executing," "ready," or "blocked."

Whenever a request finishes execution, the state of that request changes from "executing" to "completed." Upon completing execution, the request that is the immediate successor of the now "completed" request changes state from "blocked" to "ready." Once a "ready" request is assigned to a machine, the state of that request changes to "executing." The Scheduler also detects when all RCs associated with a common compound node complete execution, which triggers the Scheduler to expand the successor compound node in the WFG.

The time instant that the state of a request r transitions from "blocked" to "ready" is defined as the request's birth time and is denoted by b_r. The time instant that the state of a request transitions from "ready" to "executing" is denoted as the request's start time and is denoted by s_r. The function of the Scheduler is to define the machine assignment and start time (s_r) for each request r. The machine assignment of request r is denoted by a_r, and its value is equal to the identification number of one of the machines in the cluster.

3.4 Machine Model

The Machine Model component of Fig. 1 takes as input the request-to-machine assignments and associated start times provided by the Scheduler. The Machine Model tracks and updates an efficiency-based model for each machine in the cluster. The efficiency value for a machine depends on the aggregate CPU and memory loading due to all requests executing on the machine. At each simulation clock cycle, the Machine Model provides the Scheduler with updated efficiency values for all machines and also notifies the Scheduler of any requests that have completed execution.

The CPU and memory loading of a given machine changes with time as new requests are assigned to begin executing on the machine and existing requests complete execution on the machine. As a result, the instantaneous efficiency of a machine varies with time. Generally, the efficiency value of a machine decreases when new requests begin executing on the machine, and it increases when request(s) complete execution on that machine.

The efficiency of machine m at time t, denoted by $e(m,t)$, has a value between zero and unity. The number of CPU cycles remaining to complete execution of request r at time t is denoted by $c_r(t)$. The value of $c_r(t)$ is defined according to the following equation:

$$c_r(t) = \begin{cases} C_r, & t < s_r \\ \max\{0, c_r(t-1) - e(a_r,t)U_r\}, & t \geq s_r \end{cases} \tag{1}$$

The following discussion describes how the value of a machine's efficiency ($e(a_r,t)$ in Eq. 1) is modeled in the simulation environment. Throughout this discussion, it is understood that the efficiency value is related to a particular machine for a particular time instant. Thus, the value of efficiency is often referred to as simply e, instead of $e(m,t)$, to ease notational burden.

CPU and memory resources are the two primary factors used to characterize machines. In the machine model, the overall efficiency is defined by the product of two terms:

$$e = e_c e_m. \tag{2}$$

The terms on the right hand side of Eq. 2 are defined as the CPU efficiency and memory efficiency, respectively. The values of e_c and e_m represent the relative impact on a machine's overall efficiency due to loading of the machine's CPU and memory resources, respectively.

An idealized function for e_c has a value of unity for all CPU loadings less than the number of cores present in the machine. For CPU loading values greater than the number of cores, the idealized function for e_c decreases according to the ratio of the number of cores to the total CPU loading. In reality, overheads associated with context switching, caching effects, and other complexities that are difficult to model, would prevent the idealized efficiency curve from being realized in practice. Fig. 3 illustrates idealized and typical curves for e_c assuming a quad-core machine.

The memory resource of a machine is defined by two parameters: (1) total heap memory capacity and (2) average rate at which the machine's automatic memory management system can reclaim un-referenced heap space (i.e., garbage). For simplicity of discussion, the second parameter is assumed to be the same for all machines in the cluster. This assumption approximates a cluster configuration in which all machines implement the same virtual machine and have identical garbage collection configuration settings.

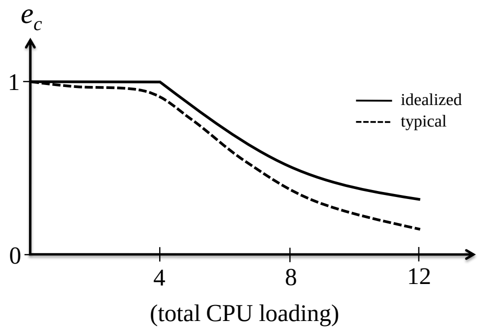


Fig. 3: Ideal and typical curves for e_c for quad-core machine.

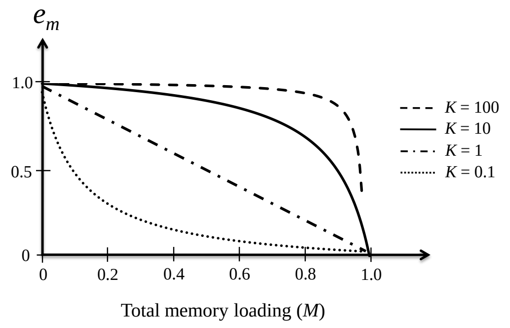


Fig. 4: Typical curves for e_m associate with Eq. 5.

In the seminal work of Hertz [4], [5], extensive empirical studies were conducted to measure how the execution time of an application is affected by the relative size of the heap memory. The general conclusion drawn from this work is that execution time is relatively constant provided that available heap space is sufficiently large. As the relative heap space is reduced, then execution time begins to increase. When the heap space is critically small, the execution time of an application can increase significantly.

In [3], a mathematical analysis is derived for the classic copying garbage collector. The basic result of the analysis is that the overhead for this garbage collector grows according to $\frac{1}{H-1}$, where H is the size of the heap normalized by the maximum reachable heap memory requirement (defined as M_r in the present paper). This expression relates to the number of garbage collections required, which clearly increases rapidly as H approaches unity. The shape of this curve associated with this function is fundamentally the same as the ones determined through extensive empirical studies in [4], [5].

To estimate how garbage collections impact the overall execution time, T, of an application, the following expression is proposed:

$$T = K + \frac{1}{H-1}. \tag{3}$$

The value of the parameter K represents the execution time when no garbage collections are performed, i.e., when the heap H is sufficiently large. The value of K is normalized in terms of the time required to perform a garbage collection.

By dividing the ideal execution time of the application, i.e., K, by the overall execution time represented by Eq. 3, an expression for e_m is derived:

$$e_m = \frac{K}{K + \frac{1}{H-1}}. \tag{4}$$

It is convenient to express e_m in terms of the memory loading on the machine, which is defined as the reciprocal of H.

$$e_m = \frac{K}{K + \frac{1}{\frac{1}{M}-1}}. \tag{5}$$

For example, a memory loading of $M = 0.5$ is equivalent to the system having a heap size that is twice as large as required by the applications(s), i.e., $M = 0.5$ in Eq. 5 is equivalent to $H = 2$ in Eq. 4. Fig. 4 illustrates e_m as a function of total memory loading for several values of K.

Fig. 5 shows a two-dimensional surface plot of $e = e_c e_m$ derived from the idealized curve for e_c depicted in Fig. 3 and the curve for e_m shown in Fig. 4 (and Eq. 5) for $K = 10$. This is the efficiency function assumed for all the machines in the cluster for the simulations conducted in Section 5. From the figure, observe that if total CPU loading of a machine exceeds its number of cores, then the efficiency of the machine decreases. Likewise, as the total memory loading of a machine increases, the efficiency of the machine decreases due to overhead associated with increased activity of the virtual machine's automatic memory management system. From the figure, observe that if a machine's CPU and memory resources are both lightly loaded, then the efficiency of the machine will be at or near its maximum value.

4. Scheduling Policies

As described in Subsection 3.3, at each simulation clock cycle the Scheduler must decide which, if any, of the "ready" requests (associated with RCs currently under consideration) should be assigned to machines and start execution. The scheduling policies considered in this paper define a prioritization by specifying the order in which "ready" requests are considered for assignment and execution on a limited number of available machines. The five scheduling polices considered are named: First Come First Serve (FCFS); First Come Last Serve (FCLS); Earliest Deadline First (EDF); Least Laxity First (LLF); and Proportional Least Laxity First (PLLF).

The FCFS and FCLS policies use the value of the time instant that a WFG arrives at the scheduler to define the priority for all requests associated with the WFG. To illustrate, assume WFG A arrives at the scheduler before WFG B. In

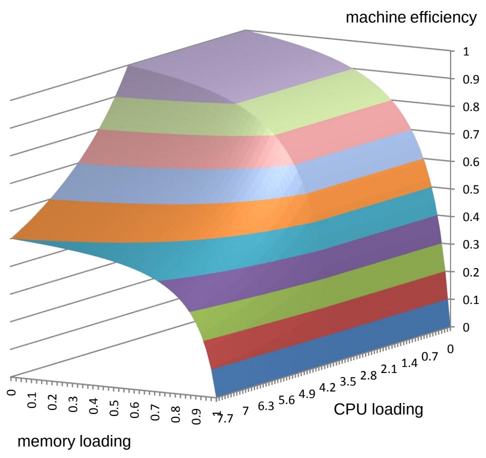


Fig. 5: Derived machine efficiency surface based on the idealized curve for e_c in Fig. 3 and the e_m curve for $K = 10$ in Fig. 4.

this case, the FCFS policy will assign a higher priority to all requests associated with WFG A (compared to the priority of all requests associated with WFG B). In contrast, for this same scenario, the FCLS policy will assign a higher priority to all requests in WFG B. The FCFS and FCLS policies are the simplest of the policies considered; they can make poor decisions because they do not consider the deadline of the WFGs in assigning priorities. The FCFS and FCLS policies are included here primarily to serve as baselines upon which the other more sophisticated policies are compared.

The EDF policy [12] prioritizes all requests of a WFG using the deadline associated with the WFG. Recall from Section 3.3 that a deadline is associated with each WFG, and this information is assumed to be known by the scheduler.

The LLF policy [12] prioritizes "ready" requests of a WFG according to their laxity, which is defined as the difference between the deadline of the WFG and the estimated finish time of the WFG. The rationale for giving requests with smaller values of laxity priority over larger values is because smaller values of laxity correspond to WFGs that are currently closer to missing their deadlines. Laxity values can be negative, and negative laxity values have priority over positive laxity values because negative laxity is an indication that the WFG deadline will likely be missed.

The PLLF policy is an enhancement of the LLF policy that uses a "proportional" laxity value to prioritize ready requests of a WFG. The proportional laxity value is defined as the laxity value divided by the ideal execution of the WFG.

Unlike the FCFS, FCLS, and EDF scheduling policies, which assign a static priority value upon the arrival of a WFG, the priority values assigned by LLF and PLLF generally vary with time. At each simulation cycle, an estimate of each WFG's finish time is first calculated. This estimated finish time is then subtracted from the WFG's

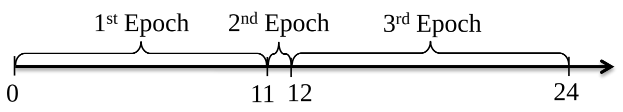


Fig. 6: Time-line illustrating the three epochs.

deadline, which yields the WFG's laxity value at that time instant.

For all policies considered in this paper, a "ready" request can only be assigned to a machine that is declared to be "available." The availability of a machine is determined using defined threshold values associated with the machine's current CPU and memory loadings. Specifically, a machine is declared to be "available" only if its CPU and memory loadings are both below defined threshold values. For all simulation results reported in the next section, the memory loading threshold used was 0.8 and the CPU loading threshold used was of 4. This pair of threshold values were shown to be superior to five other alternative pairs that were evaluated through extensive simulations: (0.8, 6), (0.8, 8), (0.5, 4), (0.5, 6), (0.5, 8).

5. Simulation Studies

Three types of WFGs are characterized: Batch, Webservice, and Interactive. The arrival times of Batch WFGs generally have daily periodicity, which distinguishes them from the other WFG types. Furthermore, Batch WFGs generally have a larger number of requests compared to the other two WFG types. The Webservice WFGs generally have more requests than Interactive WFGs. In addition to differences in arrival processes and number of requests, the different WFG types have differences related to their structure and their deadline characteristics.

The performance of the five scheduling polices are evaluated for a WFG generation scenario representing a one day period that is divided into three consecutive epochs. These three epochs are associated with WFG generation characteristics for a typical operational business day. The first epoch is from time = 0 to time = 11 hours; the second epoch is from time = 11 to time = 12 hours; and the third epoch is from time = 12 to time = 24 hours (refer to Fig. 6). During the first and third epochs, only Interactive and Webservice WFGs are generated. During the second epoch, all three types of WFGs are generated. The first and third epochs represent periods of time before and after a relatively short epoch in which Batch WFGs arrive. The start and end times of the second epoch are defined by terms of service-level agreements (SLAs) [9] related to timing of Batch WFG submission and execution. Typical terms of SLAs specify that daily Batch WFGs submitted within a specified time period will be completed by an agreed upon deadline.

Table 2: Parameter values for simulation studies.

Parameter	**Interactive WFG**	**Webservice WFG**	**Batch WFG**
Case 1: *Avg. Inter-Arrival Time (secs)	60	120	60
Case 2: *Avg. Inter-Arrival Time (secs)	10	20	60
Start – End Times (hours)	0 – 24	0 – 24	11 – 12
$^+$[Min, Max] Compound Nodes	[1, 1]	[1, 3]	[3, 5]
$^+$[Min, Max] Parallel RCs	[1, 2]	[2, 3]	[5, 20]
$^+$[Min, Max] Requests for RCs	[5, 8]	[5, 8]	[3, 8]
$^+$[Min, Max] Request Duration (secs), D_r	[1, 5]	[10, 30]	[50, 250]
$^+$[Min, Max] CPU Utilization, U_r	[0.5, 1.0]	[0.5, 1.0]	[0.5, 1.0]
$^+$[Min, Max] Heap Memory, M_r	[0.05, 0.1]	[0.05, 0.1]	[0.05, 0.15]
Parallelization Factor	2	2	2
$^+$[Min, Max] Deadline Factor	[1.1, 1.2]	[1.3, 1.5]	[1.3, 1.5]

*Poisson process. $^+$Uniform distribution.

Table 3: Statistics for simulation of Case 1 on 8 and 10 quad-core machines. “Avg. Tardy” and “Max Tardy” are in minutes.

(a) Case 1 executing on 8 quad-core machines.

Epoch	**Statistics**	**Interactive**					**Webservice**					**Batch**				
		FCFS	**FCLS**	**LLF**	**EDF**	**PLLF**	**FCFS**	**FCLS**	**LLF**	**EDF**	**PLLF**	**FCFS**	**FCLS**	**LLF**	**EDF**	**PLLF**
1st	**No. WFGs**	661					331					0				
	Tardy WFGs	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-
	Avg. Tardy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-	-	-	-	-
	Max Tardy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-	-	-	-	-
2nd	**No. WFGs**	61					30					60				
	Tardy WFGs	58	58	58	58	58	29	29	4	4	3	33	37	35	30	35
	Avg. Tardy	494.87	157.82	0.14	0.13	0.14	492.45	554.64	0.23	0.18	0.15	325.58	474.64	211.85	213.53	222.63
	Max Tardy	1037.27	1106.79	0.54	0.46	0.54	1016.33	1128.00	0.65	0.37	0.18	757.09	926.05	367.62	392.25	390.60
3rd	**No. WFGs**	721					361					0				
	Tardy WFGs	721	718	721	720	720	361	23	162	137	174	-	-	-	-	-
	Avg. Tardy	767.12	0.12	88.94	49.33	0.39	762.48	0.30	208.90	149.64	8.73	-	-	-	-	-
	Max Tardy	1036.58	1.19	363.09	290.18	2.65	1034.21	1.15	364.97	312.56	30.11	-	-	-	-	-

(b) Case 1 executing on 10 quad-core machines.

Epoch	**Statistics**	**Interactive**					**Webservice**					**Batch**				
		FCFS	**FCLS**	**LLF**	**EDF**	**PLLF**	**FCFS**	**FCLS**	**LLF**	**EDF**	**PLLF**	**FCFS**	**FCLS**	**LLF**	**EDF**	**PLLF**
1st	**No. WFGs**	661					331					0				
	Tardy WFGs	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-
	Avg. Tardy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-	-	-	-	-
	Max Tardy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-	-	-	-	-
2nd	**No. WFGs**	61					30					60				
	Tardy WFGs	58	58	58	58	58	27	26	2	2	1	27	31	25	23	25
	Avg. Tardy	387.16	0.64	0.12	0.10	0.11	411.11	444.47	0.28	0.27	0.09	223.17	353.01	78.56	78.12	82.87
	Max Tardy	811.16	3.36	0.27	0.30	0.33	807.13	873.85	0.50	0.42	0.09	550.80	675.18	144.92	168.10	160.61
3rd	**No. WFGs**	721					361					0				
	Tardy WFGs	721	714	720	716	721	361	28	136	84	154	-	-	-	-	-
	Avg. Tardy	528.72	0.11	26.12	4.88	0.23	523.88	0.29	74.18	31.67	4.72	-	-	-	-	-
	Max Tardy	810.65	0.71	136.64	80.35	1.06	808.38	1.66	141.60	88.38	15.88	-	-	-	-	-

The parameter value ranges and distributions associated with the simulation studies are summarized in Table 2. The table defines parameters related to the structural characteristics for each type of WFG, which are all assumed to be two-levels deep. Also provided in the table are CPU and memory characteristics of the requests associated with each WFG type. The parallelization factor is needed in determining a base deadline for each generated WFG; it defines the degree of parallelism assumed for executing parallel RCs associated with a common compound node. Once a base deadline is determined for a WFG, it is multiplied by the Deadline Factor (last row in the table) to define actual deadline for the WFG.

Two main cases are considered: Case 1 in which the inter-arrival times of the Interactive and Webservice WFGs are 60 secs and 120 secs, respectively; and Case 2 in which the inter-arrival times of the Interactive and Webservice WFGs are 10 secs and 20 secs. For both cases, the inter-arrival time of the Batch WFGs are assumed to be 60 secs during the second time epoch from hour 11 to hour 12; Batch WFGs do not arrive outside this one-hour interval.

The results for the simulation study of Case 1 are summarized in Table 3. The rows labeled "No. WFGs" represent the number of each type of WFG generated during each epoch. The rows labeled "Tardy WFGs" represent the number of WFGs that missed their deadline. The average and maximum tardiness statistics are calculated only for positive values of tardiness (i.e., only for WFGs that missed their deadline) and are reported in minutes.

From Table 3 observe that all policies perform well during the first epoch. This is because the loading of the system resources is so low during this time period that the choice of scheduling policy is not critical. The large Batch WFGs arrive during the second epoch, which represents a significant load on the cluster. Thus, the choice of scheduling policy has a significant impact during the second and third epochs (even though Batch WFGs do not arrive during the third epoch, requests from most Batch WFGs are still being executed well into the third epoch).

As would be expected, the FCFS and FCLS policies do not perform well under heavy loading because neither of these policies utilize deadline information in making scheduling decisions. The EDF, LLF, and PLLF policies deliver performances that are similar to each other, but the PLLF policy generally produces more desirable statistics.

Consider the simulation results for a cluster of 10 quad-core machines. In this situation, the tardiness statistics for the Batch WFGs for EDF, LLF, and PLLF are comparable to each other. However, note that the tardiness statistics during the third epoch are significantly better for PLLF than those associated with EDF and LLF. Simulations were also performed for Case 1 for 12 and 16 quad-core machines, but the numerical results are not included here because as the number of machines is increased, the performance of EDF, LLF, and PLLF become virtually the same.

Statistics associated with simulations for Case 2 (not included here) trend similarly to those of Case 1. The major difference in the two cases is that approximately 50% more machines are required for Case 2 to achieve performance that is comparable to Case 1. This is because the arrival rates for Interactive and Webservice WFGs are significantly higher in Case 2 than they are in Case 1.

6. Conclusions and Future Work

A new simulation environment is introduced for modeling the execution of workflows (WFGs) on a cluster of memory-managed multicore machines. The machine model proposed comprehends the reality that the efficiencies of memory-managed machines are impacted by the loading of their heap memories as well as the loading of their CPU resources.

The results of the simulation studies indicate that the newly proposed PLLF scheduling policy generally performs better than the other four policies evaluated. Specifically, the PLLF policy seems to handle bursts of heavy loads (the second epoch in the simulation studies) better than the other policies. Future research will be conducted to explain precisely why the PLLF policy achieves better performance for the cases considered. One hypothesis is that PLLF achieves more non-tardy Batch completion times that are close to being "just-in-time" than do the other policies.

References

[1] M. Beltrán, A. Guzmán, and J. L. Bosque, "A new cpu availability prediction model for time-shared systems," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 865–875, July 2008.

[2] Y. Zhang, W. Sun, and Y. Inoguchi, "Predicting running time of grid tasks on cpu load predictions," *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pp. 286–292, September 2006.

[3] A. W. Appel, "Garbage collection can be faster than stack allocation," *Information Processing Letters*, vol. 25, no. 4, pp. 275–279, June 1987.

[4] M. Hertz, *Quantifying and Improving the Performance of Garbage Collection.* Ph.D. Dissertation, University of Massachusetts, Amherst, 2006.

[5] M. Hertz and E. D. Berger, "Quantifying the performance of garbage collection vs. explicit memory management," *Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2005)*, October 2005.

[6] R. Jones and R. Lins, *Garbage Collection: Algorithms for Automatic Dynamic Memory Management.* John Wiley & Sons, New York, NY, 1996.

[7] H. Koide and Y. Oie, "A new task scheduling method for distributed programs that require memory management," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 941–945, 2006.

[8] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader, "Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach," *IEEE Transactions on Parallel & Distributed Systems*, vol. 18, no. 4, pp. 485–497, April 2007.

[9] D. Dyachuk and R. Deters, "Using sla context to ensure quality of service for composite services," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 865–875, July 2008.

[10] J. K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli, "Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines," *12th Heterogeneous Computing Workshop (HCW 2003), in Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, April 2003.

[11] S. H. Oh and S. M. Yang, "A modified least-laxity-first scheduling algorithm for real-time tasks," *Proceedings of the 5th International Workshop on Real-Time Computing Systems and Applications (RTCSA '98)*, pp. 31–36, October 1998.

[12] V. Salmani, M. Naghibzadeh, A. Habibi, and H. Deldari, "Quantitative comparison of job-level dynamic scheduling policies in parallel real-time systems," *Proceedings TENCON, 2006 IEEE Region 10 Conference*, November 2006.

[13] Y. Feizabadi and G. Back, "Garbage collection-aware utility accrual scheduling," *Real-Time Systems*, vol. 36, no. 1-2, pp. 3–22, July 2007.

[14] M. L. Dertouzos and A. K. l Mok, "Multiprocessor on-line scheduling of hard-real-time tasks," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, December 1989.