Speed vs. Accuracy in Simulation for I/O-Intensive Applications

Hyeonsang Eom Jeffrey K. Hollingsworth
Computer Science Department
University of Maryland
College Park, MD 20742
{hseom, hollings}@cs.umd.edu

Abstract

This paper presents a family of simulators that have been developed for data-intensive applications, and a methodology to select the most efficient one based on a user-supplied requirement for accuracy. The methodology consists of a series of tests that select an appropriate simulation based on the attributes of the application. In addition, each simulator provides two estimates of application execution time: one for the minimum expected time and the other for the maximum. We present the results of applying the strategy to existing applications and show that we can accurately simulate applications tens to hundreds of times faster than application execution time.

1. Introduction

Storage devices are frequently a major bottleneck for computer systems. To make this situation worse, the size of data stored in such systems is rapidly growing. Also, customer data storage frequently doubles in size every nine months [13], and many satellite data repositories can grow at the rate of one or two tera-bytes per day. Peta-byte level data sets will not be uncommon in a few years. To make efficient use of such complex systems, it is important to model the performance of I/O-intensive applications [1, 4, 8] and to efficiently simulate their performance to give feedback which permits improvement by changing algorithms, replacing hardware components, or changing configurations.

The performance of an application is determined by both the program to run and the machine it runs on. To evaluate the target application and machine, the application can be run directly on the real hardware. Alternatively, the code can be executed on a hardware simulator for the machine architecture. However, this execution-driven simulation is very slow. In order to increase simulation speed, it is necessary to construct an abstract version of the application that captures its execution behavior and then use a simulator modeling the target architecture at a coarser-grained level.

There are many different levels of fidelity possible when simulating I/O-intensive systems. The variety of simulation options create a speed vs. accuracy trade-off. The primary difference between simulations is the granularity of the events in the simulation. For example, each instruction is emulated in an instruction-level simulation while instructions grouped in basic blocks are executed in an execution-driven simulation [5]. At an even coarser level, an event could be a complex data transfer operation corresponding to millions of machine instructions. A second source of changing fidelity comes from event dependency: some events should occur after other events. For example, an event representing a receive operation for a message needs to be processed after an event for the corresponding send operation. For accurate performance prediction, it is important to preserve this dependency; yet to preserve it, events need to be individually processed in the correct order.

This paper presents a set of event-based simulation options (simulators) for data-intensive applications, and a methodology to select the most appropriate one based on a user-supplied requirement for accuracy. The methodology consists of a series of tests that select the least expensive simulation that provides desired accuracy based on the attributes of the application and its execution. The main advantage of using the methodology is that, given multiple simulation options, it determines if the result of the less expensive option is sufficiently accurate, and uses it. Our technique is primarily intended to help application developers choose between different application options or select application resources to request (i.e. to manage their allocation at a supercomputing center) although it could be used to help configure new systems or to design new hardware. Currently our method is semi-automated with many of the tests conducted automatically by processing application event data.

2. Execution model

The execution of I/O-intensive applications is modeled by a series of events performed on a collection of items called a data block. Events represent read, write, send, receive, and compute operations for each data block[1]. The execution of events can depend on that of other events. For example, a message send event must proceed the corresponding receive event. In addition, multiple events can depend on a single event. The specific events and the relationship of the events for an application execution are described via dynamic Work-Flow Graphs (WFGs). In a WFG, nodes correspond to events and edges represent their dependency. The abstract model represents

[1] Initially, our target I/O-intensive applications run on message passing systems.

the major activities that determine the execution time of data-intensive applications and thus serves as a natural representation for simulators.

Figure 1 illustrates three WFGs that represent the execution of a simple application on two processors and two disks. Disk1 and Disk2 are attached to Processor1 and Processor2, respectively. The vertical lines, one for each of the processors and disks, represent the processing of events by a device. In this graph, three data blocks are read from the disks, and subsequently used in either local or remote computation. Although we have placed the send event before the compute event on Processor1, there is no ordering constraint that forces the events to be processed in this order.

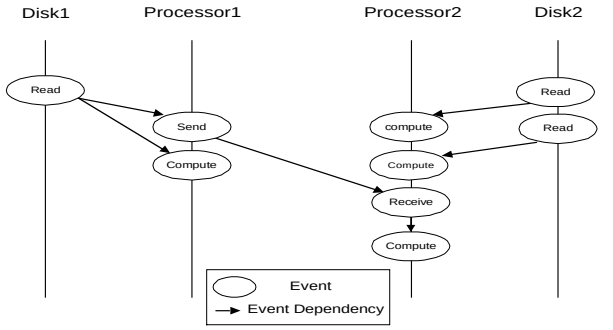

Figure 1 An Illustration of Work Flow Graphs.

WFGs are generated by executing instrumented programs; the programs can be either real ones or application emulators. Application emulators are kernels of the target applications, whose execution behavior represents that of a whole application or class of applications. The emulators model the computation and data-access patterns of the applications in a controlled manner. They efficiently provide a representation of the application's dynamic behavior. More information about application emulators can be found in [20].

Our hardware simulators perform discrete-event simulation by processing the events of WFG produced by applications or emulators. The simulators maintain one global simulation clock and a resource clock for each major hardware component. In our system, a resource clock is associated with each of the disks and processors. When an event is processed, the resource clocks are updated. The global clock tracks the largest resource-clock value. The processor clocks are also used as network clocks since we assume that protocol processing is performed by the processors. We ignore time of flight for messages since protocol processing costs dominate hardware latency on most current clusters.

3. Simulation options

We now describe the different simulation options and their relative costs. The simulation options we consider are all discrete-event based, but differ in terms of the granularity of simulated events and the fidelity of hardware data used in the simulation. The options are divided into two main categories: fine-grained and coarse-grained.

Coarse-grained simulation includes two categories: dependency-preserving and non-dependency-preserving. In dependency-preserving mode, events are processed one by one according to their dependency order. However, in resource-event simulation, all events for each resource are aggregated as a single "resource event." In essence, we simply add up the time of the requests for each resource and report the largest sum. Although extremely coarse, this simulation option often provides useful insight into the minimum execution time of an application.

There are three sub-methods of dependency-preserving simulation: macro-event-round-robin, macro-event queue, and micro-event queue. The first two are efficient, but somewhat inaccurate versions of the third. Micro-event-queue-based simulation precisely orders all events using a global event-processing queue. The macro-event-queue option coarsens the granularity of events by treating each WFG as a "macro-event," and determines the processing order of WFGs using a global WFG queue. On the other hand, the macro-event-round-robin method decides the order by selecting WFGs in a round-robin fashion across all processors without using a queue.

Our fine-grained simulation models the following execution phenomena: contention/congestion for shared resources, disk seek time, variation in I/O time (e.g., due to disk layout), CPU involvement in I/O, and event-completion delay due to platform-dependent artifacts.

4. Selecting the simulation options

We have developed a taxonomy of I/O simulation that allows selecting the most efficient simulation method to meet a user-supplied target error bound. The key idea is to start with coarse-grained simulation that tends to have a large error range and repeatedly try more sophisticated techniques until the error bound has been reduced to within the user-supplied threshold. Figure 2 shows the taxonomy. Ellipses in the figure represent tests where a decision is made. Rectangles depict running a specific simulation technique and testing whether the technique meets the accuracy requirement. Based on the results of these tests, new simulation options are run, and their acc-

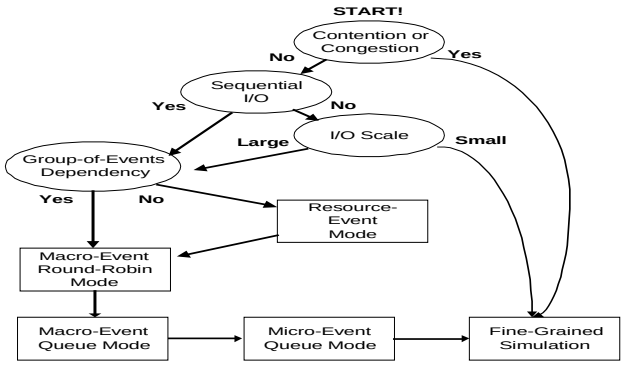

Figure 2 Decision Tree to Select Simulation Method.

uracy evaluated.

Initially, the WFGs to be simulated are tested to decide whether it is possible to predict execution time using a coarse-grained option. These validity tests check whether there are any critical application or platform performance factors that would be lost with our coarse-grained options. There are three such tests: contention/congestion, sequential I/O, and I/O scale.

Contention or Congestion Test: checks whether there will likely be runtime contention or congestion for resources. When testing for contention or congestion, we consider only limited networking capacity[2]. Communication end-point congestion [19] and link contention typically occur when the communication rate is high and one or more processors are involved in a many-to-one or one-to-many communication. We determine whether there is likely to be communication contention or congestion by running two short communication-only programs with the same communication parameters as the application to be simulated. The first test program measures communication time including contention or congestion by sending data to/from servers. The second test program measures communication time in the absence of contention or congestion, using a simple ping-pong benchmark. The effect of communication contention or congestion is the difference between the per-block communication times of the two test programs times the maximum number of data blocks communicated per processor. The parameters of the two communication programs for the contention/congestion test are set as follows:

Fan-out: indicates how many processors receive a data block, and is set to the maximum number of processors that any processor communicates with during the execution of the target application.

Communication rate: is the per-processor block-sending rate, and is computed as:

$$FO * N_{local\_disk} \Big/ T_{IO} ,$$

where *FO* is the maximum fan-out value of the execution of the application, N_{local_disk} is the maximum number of disks on any one node used during the execution, and T_{IO} is the mean per-block I/O time. *FO* and N_{local_disk} are computed from the input WFGs of the application. T_{IO} is measured by running a simple I/O benchmark program parameterized by the size of data blocks, number of local disks used, and disk-seek method (either sequential or completely random). To measure T_{IO}, the block-size parameter is set to the mean block size used by the application, and the other two parameters are set to N_{local_disk} (defined above) and sequential seek respectively.

Data-block size: is the data-block size of the test programs, and is set to the mean data-block size used by the application.

[2] We found limited networking capacity to be the main source of contention or congestion in our experiments.

Sequential I/O Test: evaluates if the data-block access is (fully) sequential for each disk during execution. The information is extracted from the WFG logs by examining the file offsets of adjacent disk requests. If I/O is not completely sequential, then it is necessary to decide whether to simulate disk I/O seek time.

I/O Scale Test: determines if the disk seek time is large enough to require incorporation into the simulation. A parameterized test program that performs I/O operations of a similar size to the application being simulated is run. The test program is run twice, once using totally sequential I/O requests and once with completely random I/O requests. If the additional time due to random seeking is larger than the target error, then a fine-grained simulation that includes seek time needs to be performed.

Group-of-Events-Dependency Test: decides if the non-dependency-preserving resource-event simulation can be selected. The test checks whether there are group-of-events dependencies. Group-of-events dependencies occur when one group of events must finish before any event in the next group can be started. For example, in some database applications, the results of one query must be processed before the next query can be started. This type of dependency information is explicitly represented in WFGs.

5. Error bound of simulations

Our simulators produce two values for the expected running time of the application: an optimistic value, T_{pl}, which corresponds to the shortest running time and T_{ph} which is a pessimistic estimate of the longest running time. If the maximum error is within the target error, the simulator is selected; otherwise, the next-most-detailed simulation is run. We now explain each simulator in detail.

5.1 Resource-event simulation

Resource-event simulation processes all events of each resource as a single "resource event" without considering dependencies between events. It is the simplest of our techniques. The error-bound test that determines whether the simulation can be used checks to see if a single resource dominates execution time and thus hides the execution time of other activities. If a single resource dominates the execution time, we can greatly reduce the fidelity of the other parts of the simulation without reducing accuracy. We regard the resource with the maximum clock value as the candidate for the dominant one, and set T_{pl} to this value.

$$T_{pl} = \max_{R(esource)} (T^R) ,$$

where T^R is the final clock value of a resource R. However, it is possible that activities of resources other than the dominant resource are not hidden by the dominant one because of order constraints. Based on this, T_{ph} is computed as:

$$T_{ph} = \begin{cases} \sum_{p(rocessor)} (T^{p} + E^{p}_{Overlap}) + \max_{d(isk)} (T^{d} + E^{d}), & w/\ comm. \\ \max_{\substack{p(rocessor)\& \\ local\_d(isk)}} (T^{p} + E^{p}_{Overlap} + T^{d} + E^{d}), & wo/\ comm., \end{cases}$$

where $E^{p}_{Overlap}$ is the total maximum error introduced for a processor *p*, and E^{d} is the error for a disk *d*. In the worst case, all events of a resource might need to be processed after all events of another resource (i.e. resource event by resource event) for all resources. Hence, the simplest, highly conservative, estimate of execution time could be the sum of the values of all resource clocks and the total maximum error introduced for the resources. However, this estimation is too conservative because operations of one disk are performed in parallel with those of other disks. Therefore, T_{ph} is the sum of the worst-case disk time plus the worst-case processor time, where the processor time is computed as the sum of the values of all processor clocks and the associated non-overlapping error for the processors, if there is inter-processor communication. Otherwise, T_{ph} is the maximum of the per-processor sums of the worst-case processor and disk times across all processors.

When computing the maximum error introduced for disks, E^{d}, we need to consider Random I/O and I/O Variation. Therefore, E^{d} is computed for a disk *d* as:

$$E^{d} = E^{d}_{Rand\_IO} + E^{d}_{IO\_Var}$$

The random I/O effect represents the error introduced by not simulating seek times for the disks. The difference between per-block I/O time using sequential seek and the time using Random Seek, T_{Rand_Seek}, computed in the I/O scale test is used to quantify this phenomenon. The maximum error introduced for a disk *d*, is,

$$E^{d}_{Rand\_IO} = T_{Rand\_Seek} * N^{d}_{IO},$$

where N^{d}_{IO} is the total number of I/O operations for the disk. The I/O variation effect indicates the variation of I/O time among processors, and is measured as the maximum Difference in per-block I/O time between any two processors, T_{IO_Diff}. The maximum error due to this phenomenon for a disk *d*, is,

$$E^{d}_{IO\_Var} = T_{IO\_Diff} * N^{d}_{IO}$$

For the maximum error introduced for processors, $E^{p}_{Overlap}$, we need to consider computation and I/O Overlap. The overlapping effect represents CPU involvement in I/O, and the worst-case per-block effect of this phenomenon is computed as the per-block wall-clock time spent in I/O routines, T_{IO_CPU}. The total error caused by this phenomenon for a processor *p*, is,

$$E^{p}_{Overlap} = T_{IO\_CPU} * N^{p}_{IO},$$

where N^{p}_{IO} is the total number of I/O operations for the processor.

5.2 Macro-event-round-robin simulation

Macro-event-round-robin simulation processes the events of a single WFG atomically as a "macro-event" while selecting the next WFG to process in a round-robin fashion across disks. The error-bound computation takes into account communication Balance, Fan-Out & the number of disks, Event-completion Delay, Random I/O, I/O Variation, and computation & I/O Overlap. T_{pl} and T_{ph} are computed as follows:

$$T_{pl} = T_{p} - E_{Balance} - E_{FO}$$

$$T_{ph} = T_{p} + E_{Event\_Delay} + E_{Rand\_IO} + E_{IO\_Var} + E_{Overlap},$$

where T_{p} is the final global clock value at the completion of the simulation.

The unbalanced-communication error, $E_{Balance}$, represents the maximum difference of the accumulated communication time between any two processors, and accounts for the simulation error caused by round-robin processing of macro-events.

$$E_{Balance} = \max_{p(rocessor)} (T^{p}_{Comm}) - \min_{p(rocessor)} (T^{p}_{Comm}),$$

where T^{p}_{Comm} is the total protocol-processing time (processor communication time) on a processor *p*. When I/O and communication are not overlapped and there are the same number of I/O operations on different disks, the completion times can be different due to the non-overlapping between I/O and communication. The maximum difference in the completion time between any two disks is $E_{Balance}$. Since these completion times can vary across disks, and an I/O event for a disk starts being processed at the completion time of the previous I/O event for the same disk, I/O events can be processed out of order when the next I/O event to process is selected in a round-robin fashion across disks. In the worst case, an I/O event is processed $E_{Balance}$ earlier than it should be. When an I/O operation completes, this time advances the clocks on all of the processors that receive that block (since message passing advances a resource clock to the larger of the sender or receiver's clocks). Since all events of a WFG are processed atomically, the completion times of I/O events are immediately propagated to the receivers without allowing any other events to be processed between the I/O events and receive events. Therefore, in the worst case, an I/O event that was processed earlier by $E_{Balance}$ would prevent other events from being processed during the "waiting" time on the corresponding receiver, resulting in a delay in processing those events by that amount.

The fan-out error, E_{FO}, indicates the simulation error caused by increasing event granularity by processing the events of WFGs atomically, and is computed as:

$$E_{FO} = \begin{cases} E_{FOl}, & if\ I/O\ is\ overlapped\ with\ comm. \\ E_{FOh}, & otherwise \end{cases}$$

If I/O is completely overlapped with communication, the fan-out error is,

$$E_{FOl} = FO * T_{comm},$$

where *FO* is the maximum fan-out, and T_{comm} is the per-block communication time (contention/congestion-free as computed by the contention/congestion test). The extra potential error results from the fact that all events of a WFG are processed one after another without being interleaved by processing of events of any other WFG. This could result in the loss of interleaving opportunities for event processing, thereby causing a delay in processing of some events. Since all events of a WFG are processed together in macro-event-queue simulation, this results in advancing the processor clock of each node receiving the data block for the WFG without allowing any other events to be processed while that node would be "waiting." If some events that should have been processed during the time are processed later, simulation time can be increased by that amount. Therefore, we subtract E_{FOl} from T_p when computing the lower bound.

On the other hand, if I/O can not be completely overlapped with communication, the fan-out error, is,

$$E_{FOh} = (FO + N_{disk}) * T_{comm},$$

where N_{disk} is the total number of disks used, because the simulation error which is equal to $N_{disk} * T_{comm}$, can be added to the fan-out error. Since all events of a WFG are processed atomically, some communication events can be processed earlier than if events from other WFGs were interleaved. When this happens, the communication events are processed earlier than they should have been processed. In the worst case, N_{disk} communication events, one from each disk, can be processed earlier than they should be processed. In this case, the completion of an I/O event could be incorrectly delayed by $N_{disk} * T_{comm}$. Subsequently this effect would be propagated to all receivers of the data block, contributing to simulation error by that amount.

Event-completion delay, random I/O, I/O variation, and computation & I/O overlap constitute the error sources that can cause an underestimation of application execution when using our micro-event or macro-event simulations.

The simulation error, E_{Event_Delay}, due to event-completion-delay represents the effect of delay in completion of some operations until completion of dependent operations. The delay is measured as the maximum difference in simulation time value between any two processors.

$$E_{Event\_Delay} = \max_{p(rocessor)}(T^p) - \min_{p(rocessor)}(T^p)$$

The simulation errors, E_{Rand_IO}, E_{IO_Var}, and $E_{Overlap}$, due to random I/O, I/O variation, and computation & I/O overlap are computed using the equations for the effects of the phenomena described in Section 5.1 for resource-event simulation as follows:

$$E_{Rand\_IO} = \max_{d(isk)}(E^d_{Rand\_IO}); E_{IO\_Var} = \max_{d(isk)}(E^d_{IO\_Var});$$

$$E_{Overlap} = \max_{p(rocessor)}(E^p_{Overlap})$$

5.3 Macro-event-queue simulation

Macro-event-queue simulation processes WFGs as "macro-events" using a global WFG queue. The error ranges consider the same error sources that affect the accuracy of macro-event-round-robin simulation except unbalanced communication. This error source is not considered because communication imbalance does not introduce any simulation error due to in-order processing of macro-events as explained in Section 5.2. T_{pl} and T_{ph} are computed as follows:

$$T_{pl} = T_p - E_{FO}$$

$$T_{ph} = T_p + E_{Event\_Delay} + E_{Rand\_IO} + E_{IO\_Var} + E_{Overlap},$$

where T_p is the result of the simulation.

5.4 Micro-event-queue simulation

Micro-event-queue simulation processes WFG events as "micro-events" using a global event queue. Computing the error range of this simulation considers event-completion delay, random I/O, I/O variation, and computation & I/O overlap as the error sources that determine the accuracy of the simulation. T_{pl} and T_{ph} are computed as follows:

$$T_{pl} = T_p$$

$$T_{ph} = T_p + E_{Event\_Delay} + E_{Rand\_IO} + E_{IO\_Var} + E_{Overlap},$$

where T_p is the result of the simulation.

5.5 Fine-grained simulation

Fine-grained simulation uses a global event queue to process fine-grained events that represent detailed-level application activities such as movement of data between components of devices. The accuracy of the simulation is not affected by any of the error sources that influence coarse-grained simulation: unbalanced communication, fan-out, the number of disks, event-completion delay, random I/O, I/O variation, and computation & I/O overlap. Other error sources that affect the accuracy include limitations in the system's ability to overlap I/O & communication. Also, contention and congestion for shared resources are only modeled to a limited extent. We assume that the amount of error due to the other sources is also zero because for the type of applications we are interested in, communication and I/O operations are performed on large blocks of data, and thereby these effects tend to be very small.

6. Examples of applying the strategy

This section presents some examples to validate the accuracy of our strategy, and demonstrate its utility to efficiently predict the execution time of I/O-intensive applications. We explain how to apply the strategy to two data-intensive applications: Titan and the Virtual Microscope. Titan [4] is a parallel shared-nothing database server that stores satellite data, and processes queries for

the data. The Virtual Microscope [8] is a server that processes queries from multiple clients for digitized images of visual microscope slides.

6.1 Validation of error bounds

To demonstrate the correctness of the error bounds used in our strategy, we show that actual measurements for Titan and the Virtual Microscope are within the error ranges of the simulation options. We performed our validation experiments on 12 nodes of an IBM SP2 with four disks attached to each node.

Figure 3 shows the results of simulation for Titan and the Virtual Microscope. The columns show the predicted time, actual (percent) error of the predicted time with respect to the corresponding actual measurement, and (low and high) raw and combined error bounds with their error ranges for each option. The lower and upper bounds for each option are computed based on the predicted time and maximum error shown in Section 5. The combined lower and upper bounds are the maximum of all available low bounds and the minimum of all high bounds, respectively. This allows us to produce a hybrid range based on the composition of the simulations that have been performed. The combined error bounds are shown in the last three columns of Figure 3.

The input parameter sets commonly include data-block size, the location of input-data files, the total volume of data to process, the structure of 2D processor mesh (including the total number of processors), the total number of disks per processor, the location and size of a query window, the number of queries, and per-block computation time. The additional parameters for the Virtual Microscope are the number and size of slides. In all cases, the measured execution times, 336.8 seconds for Titan and 683.4 seconds for the Virtual Microscope, fall within the error bounds shown in Figure 3. Although we present the results for the resource-event option in Figure 3 for Titan, the group-of-events dependency test indicates that the technique produces unreliable results, so we don't show the error bounds for this case. The raw and combined bounds are the same in Figure 3 for the Virtual Microscope because the results of all coarse-grained options are the same in the Virtual Microscope case. They are identical since there is neither random I/O, group-of-events dependencies, nor communication.

6.2 Application of the strategy

In this section, we present a simulation study to show the process of selecting an appropriate simulation option using our methodology. For each of the applications, we use a larger configuration than in the previous validation study: 12 TB and 1,200 nodes for Titan, and 1.2 TB (and 12 nodes) for the Virtual Microscope. In all cases, we set the target error bound to 20%.

Figure 4 shows the results of simulation options that can be used for the version of Titan in the larger configuration. It presents the predicted time, (low and high) raw and combined error bounds with their error ranges. The simulation results for the resource-event option in this figure are also unusable because there are group-of-events dependencies.

The macro-event-round-robin simulator is first tried. In this simulation mode, we first check whether I/O is overlapped with communication. I/O is not overlapped with communication in the target platform since we assume that the platform is a scaled-up version of the SP2 used in the previous section. Consequently, the results of running the macro-event-round-robin simulator are checked to see whether the option is sufficiently accurate. Since the desired error is 20%, this test fails because the current simulation error range, 58%, is larger than the target error.

Next the macro-event-queue-based simulator is run. The low and high estimates of execution time provided by this option are checked to determine whether its error range is within the target error. Since the error range, 6%, is less than the target error, the result of macro-event-queue-based simulation is used as the final predicted time (boldfaced in the table). The relative simulation error, 95.1%, is larger than that of the coarser-grained macro-event-round-robin option because the lower bound of this option is much smaller than that of the macro-event-

Titan Simulation Option	Predicted Time (secs)	Actual Error (%)	Range of Time					
			Raw			Combined		
	Measured Time: 336.8		Low	High	Error (%)	Low	High	Error (%)
Resource Event	185.6	44.9	N/A	N/A	N/A	N/A	N/A	N/A
Macro-Event RR	355.8	5.6	335.9	404.4	20.4	335.9	404.4	20.4
Macro-Event Queue	302.4	10.2	302.2	351.0	16.1	335.9	351.0	4.5
Micro-Event Queue	302.3	10.2	302.3	350.9	16.1	335.9	350.9	4.5
Virtual Microscope Simulation Option	**Predicted Time (secs)**	**Actual Error (%)**	**Range of Time**					
			Raw			Combined		
	Measured Time: 683.4		Low	High	Error (%)	Low	High	Error (%)
Resource Event	619.2	9.4	619.2	705.0	13.9	619.2	705.0	13.9
Macro-Event RR	619.2	9.4	619.2	705.0	13.9	619.2	705.0	13.9
Macro-Event Queue	619.2	9.4	619.2	705.0	13.9	619.2	705.0	13.9
Micro-Event Queue	619.2	9.4	619.2	705.0	13.9	619.2	705.0	13.9

Figure 3 Simulation Results for the Titan and Virtual Microscope Emulators with 12 GB Data.

Simulation Option	Predicted Time (secs)	Range of Time					
		Raw			Combined		
		Low	High	Error (%)	Low	High	Error (%)
Resource Event	2,364.5	N/A	N/A	N/A	N/A	N/A	N/A
Macro-Event RR	5,177.9	4,844.8	7,660.3	58.1	4,844.8	7,660.3	58.1
Macro-Event Queue	**2,650.7**	**2,630.7**	**5,133.1**	**95.1**	**4,844.8**	**5,133.1**	**6.0**
Micro-Event Queue	2,650.6	2,650.6	5,133.0	93.7	4,844.8	5,133.0	5.9

Figure 4 Simulation Results for Titan with 12 TB Data on 1,200 Nodes.

round-robin option. However, the absolute simulation error, the difference between the lower and upper bounds of this option, is smaller than that of the macro-event-round-robin option. We show the results from the micro-event-queue-based simulator for illustrative purposes even though there is no need to run this simulator based on the achieved error bounds with the previous technique.

Figure 5 shows simulation time in seconds and the speedup with respect to the corresponding predicted time. The speedup factor indicates how much faster it is to simulate an application rather than running it. In this example, the final simulation option used takes less than five minutes as shown in Figure 5. By applying the strategy in choosing the best option, we can avoid using the micro-event-queue option that takes more than twenty minutes, but that provides a result of the similar fidelity.

	Sim. Time	Speedup wrt Pred. Time
Resource Event	32.8	72.1
Macro-Event RR	177.3	29.2
Macro-Event Queue	**275.8**	**9.6**
Micro-Event Queue	1,343.8	2.0

Figure 5 Titan with 12 TB Data on 1,200 Nodes.

Figure 6 shows the predicted time, low and high error bounds and error ranges of each simulation option for the Virtual Microscope with 1.2 TB data set on 12 nodes (100 GB data per node). For all four options shown, the predicted execution time is just over 18 hours (66,126 seconds). The simulation's predicted error bound is 14%.

In this example, there is neither network contention /congestion, nor random I/O. Also, there is no group-of-events dependency. As a result, the resource-event simulator is run. The error range of that option is 14%, less than the target error; therefore, the option is selected as the best one. The other coarse-grained options show the same simulation result as that of this option, and thus provide no additional fidelity. The results from the coarse-grained simulators other than the chosen one are also shown for illustrative purposes; there is no need to run those simulators based on the achieved error bounds with the resource-event technique.

Figure 7 presents simulation time in seconds and the ratio of the simulation time to the predicted execution time. The speedup factor for the resource-event option (the one selected by our methodology) is 917. This indicates that we are able to predict the execution time of a program over 900 times faster than running the application. Also, our strategy selects the resource-event option that takes about one minute rather than other coarse-grained options that can take more than five minutes. The results in this example show the benefit of using our operation-selection strategy: when the most inexpensive simulation option is sufficiently accurate, the strategy identifies that fact and avoids running the more expensive ones.

	Sim. Time	Speedup wrt Pred. Time
Resource Event	**70.5**	**917.1**
Macro-Event RR	175.0	377.9
Macro-Event Queue	260.7	253.6
Micro-Event Queue	325.2	203.3

Figure 7 Virtual Microscope with 1.2 TB Data.

7. Related work

Many simulation studies have addressed trading speed vs. detail. SimOS [17], a complete computer system simulator, provides three interchangeable simulation modes: positioning, rough-characterization, and accurate modes. It allows selecting among the modes dynamically so that it can simulate only interesting sections of execution in detail; however, it doesn't provide a detailed strategy to change its simulation mode in order to achieve the best performance with respect to an error requirement. Other simulation systems [6, 9, 12] that permit multi-level simulation have also been developed. None of these systems allow selecting the most efficient level of simulation that meets a target error bound. However, these systems explicitly consider the effects of system activities such as

Simulation Option	Predicted Time (secs)	Range of Time					
		Raw			Combined		
		Low	High	Error (%)	Low	High	Error (%)
Resource Event	**66,126.2**	**66,126.2**	**75,293.1**	**13.9**	**66,126.2**	**75,293.1**	**13.9**
Macro-Event RR	66,126.2	66,126.2	75,293.1	13.9	66,126.2	75,293.1	13.9
Macro-Event Queue	66,126.2	66,126.2	75,293.1	13.9	66,126.2	75,293.1	13.9
Micro-Event Queue	66,126.2	66,126.2	75,293.1	13.9	66,126.2	75,293.1	13.9

Figure 6 Simulation Results for the Virtual Microscope with 1.2 TB Data.

caching and buffering while our system doesn't.

A parallel simulation study [3] using Wisconsin Wind Tunnel [16] showed various performance trade-offs for six different network simulation models. However, they did not provide a strategy to select the best model. On the other hand, other studies [2, 16] focused on providing both efficient and accurate simulators by direct execution and/or parallel simulation.

POEMS (Performance Oriented End-to-end Modeling System) [7] is an integrated end-to-end performance modeling system that allows different target components to be modeled at multiple levels of detail (multi-scale) by different paradigms (analytic modeling, simulation, and actual system execution). It is similar to our simulation system in that it provides multiple options that allow trading simulation speed vs. simulation accuracy; however, it does not estimate error bounds for each option, which makes it impossible to provide an option selection strategy with respect to a target error. COMPASS (COMponent-based Parallel System Simulator) [2] is a direct execution-driven, parallel simulator used for detailed simulations within the POEM project.

A time warp simulation [10] asynchronously advances the clocks of Logical Processes (LPs) for events in timestamp order while LPs communicate via messages. It provides a *rollback* for out-of-order messages to restructure the simulation to process events for the messages in the correct timestamp order. In our system, events can be processed out of order due to event aggregation; however, we quantify the error range resulting from the aggregation rather than correcting it by performing an expensive rollback. In contrast to our approach, a number of technologies have been developed to reduce the cost of rollback. Efficient checkpointing [14, 18] schemes that save LP state, and GVT estimation and fossil collection [15], have been devised to improve simulation performance and decrease the amount of required memory. A recent survey on languages and libraries of Parallel Discrete-Event Simulation (PDES) can be found in [11].

8. Conclusion and future directions

We described a set of event-based simulation options for I/O-intensive applications and showed a time vs. accuracy trade-off depending on the level of event aggregation. We presented a strategy that allows selecting the most efficient simulation option while meeting an error bound for those options, and demonstrated its effectiveness for two existing data-intensive applications.

A future direction of this research is to dynamically apply the option-selection strategy so that the best option can be chosen at runtime. To do this, input events need to be generated online. Also, a mechanism to accurately transfer information of simulation status between different simulators at option-change points needs to be implemented. Once the best option is selected, it can be used to the end of execution, or the strategy can periodically applied to adapt simulation method according to changes of the attributes of the target application and its execution.

Acknowledgements

We thank one of the anonymous reviewers for their extremely detailed and thoughtful comments.

References

1. R. Agrawal and J. Shafer, "Parallel Mining of Association Rules," *IEEE Transactions on Knowledge and Data Engineering*, **8**(6), 1996, pp. 962-969.
2. R. Bagrodia, E. Deelman, S. Docy, and T. Phan, "Performance Prediction of Large Parallel Applications Using Parallel Simulations," *ACM PPoPP*, May 1999, Atlanta, GA, pp. 151-161.
3. D. C. Burger and D. A. Wood, "Accuracy vs. Performance in Parallel Simulation of Interconnection Network," *9th ACM/IEEE IPPS*, April 1995, Santa Barbara, CA, pp. 22-31.
4. C. Chang, *et al*., "Titan: A High-Performance Remote-Sensing Database," *13th ICDE*, April 1997, United Kingdom, pp. 375-384.
5. R. G. Covington, *et al*., "The Efficient Simulation of Parallel Computer Systems," *International Journal in Computer Simulation*, **1**, 1991, pp. 31-58.
6. H. Davis, S. R. Goldschmidt, and J. Hennessy, "Multiprocessor Simulation and Tracing Using Tango," *1991 ICPP*, August 1991, St. Charles, IL, pp. 99-107.
7. E. Deelman, *et al*., "POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational System," *International Workshop on Software and Performance*, October 1998, Santa Fe, NM, pp. 18-30.
8. R. Ferreira, *et al*., "The Virtual Microscope," *1997 AMIA Annual Fall Symposium*, October 1997, Nashville, TN, pp. 449-453.
9. R. S. Francis and I. D. Mathieson, "Compiler-Integrated Multiprocessor Simulation," *International Journal in Computer Simulation*, **1**(2), 1991, pp. 169-188.
10. D. Jefferson, "Virtual Time," *ACM TPLS*, **7**(3), 1985, pp. 405-425.
11. Y. H. Low, *et al*., "Survey of Language and Runtime Libraries for Parallel Discrete-Event Simulation," *The Journal of the Society for Computer Simulation*, **72**(3), 1999, pp. 170-186.
12. P. S. Magnusson, *et al.*, "SimICS/sun4m: A Virtual Workstation," *Usenix 1998 Annual Technical Conference*, June 15-18, 1998, New Orleans, LA, pp. 119-130.
13. G. Papadopolous, *The Future of Computing*, 1997, Unpublished talk at *NOW Workshop*.
14. R. Radhakrishnan, N. Abu-Ghazaleh, M. Chetlur, and P. A. Wilsey, "On-line Configuration of a Time Warp Parallel Discrete Event Simulator," *1998 ICPP*, August 1998, Minneapolis, MN, pp. 28-35.
15. P. L. Reiher, "Parallel Simulation Using the Time Warp Operating System," *1990 Winter Simulation Conference*, December 1990, New Orleans, LA, pp. 38-45.
16. S. K. Reinhardt, J. R. Larus, and D. A. Wood, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," *ACM SIGMETIRCS*, May 1993, Santa Clara, CA, pp. 46-60.
17. M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *ACM Transactions on Modeling and Computer Simulation*, **7**(1), 1997, pp. 78-103.
18. J. S. Steinman, "Incremental state saving in SPEEDES using C++," *1993 Winter Simulation Conference*, December 13 - 16, 1993, Los Angeles, CA, pp. 687 - 696.
19. M. Uysal, A. Acharya, R. Bennett, and J. Saltz, "A Customizable Simulator for Workstation Networks," *11th IPPS*, April 1997, Geneva, Switzerland, pp. 249-254.
20. M. Uysal, T. M. Kurc, A. Sussman, and J. Saltz, "A Performance Prediction Framework for Data Intensive Applications on Large Scale Parallel Machines," *4th Workshop on Language, Compiler and Run-Time Systems for Scalable Computers*, May 1998, Pittsburgh, PA, pp. 243 -258.