Computer Cluster and Grid Simulator

I. Grudenic and N. Bogunovic
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of electrical engineering and computing, University of Zagreb
Unska 3, Zagreb, Croatia
Phone: 01-6129-999 int. 548 Fax: 01-6129-653 E-mail: igor.grudenic@fer.hr

Abstract – Evaluation of scheduling algorithms for distributed environments is a complex task that involves simulation of computation resources, interconnection network, users and generated workload. Multiple discrete event based simulators are developed that tackle different aspects of distributed systems in order to ease that task. In this paper we address the simulation of computer clusters that are the most popular distributed system architecture. Fundamentals of the available simulators are given and the extendable architecture for the new simulator is proposed. Three-phase discrete event simulation is recognized as the simulation technique of choice, and the behavior of the key components is modeled using that technique. Implementation details targeted to object oriented environments are pointed out.

I. INTRODUCTION

Computer clusters are a preferred distributed system architecture type because of the high price/computing performance ratio. They constitute 82% of the top 500 supercomputers [1], with the rest being massively parallel processors (MPP) and constellation systems.

The most important aspect of the computer cluster is its performance. Performance can be defined in a variety of metrics [2], and the job of the cluster scheduler is to optimize the behavior of the cluster according to the target metric.

Cluster scheduling in general is known to be an NP complete problem [3]. This complexity makes exhaustive schedule searching and application of the complex heuristics infeasible. In large computing clusters event rate is very high and the applied scheduling algorithm must provide a matching decision rate [4][5].

In order to cover different metrics and workload types different scheduling algorithms are developed. In the development phase these are simulated using synthetic workload traces. In this paper we address the simulation of the cluster scheduling algorithms starting from the underlying discrete event simulation systems and following with the visualization of the results. Additionally we propose the architecture for the specified system together with the implementation details. This architecture is extendable and can easily apply to architectures like MPP and constellation systems.

General architecture of the cluster systems that is a basis for the simulation is described in section II. Available discrete event simulation techniques are described in section III. Section IV presents a short overview of the existing cluster and grid simulation tools. Architecture, simulation and implementation details of the proposed cluster simulator are given in section V.

II. CLUSTER ARCHITECTURE

Computer cluster is a group of interconnected computers that are usually connected using fast local area network. From the administrative point of view they are centrally managed at the designated frontend node, while other computers are referenced as computing nodes.

Frontend node is used for job submission and resource scheduling. Some cluster suits [6] allow submission of jobs at any cluster node but scheduling is still performed at the dedicated cluster node. Cluster scheduler responds to two types of events in the system. User generated events represent job submissions and cancellations as well as the resource reservation. System generated events include job status information and resource availability. System generated events are collected by the resource manager component that is distributed across all cluster nodes. Using the information collected from the users and the resource manager, cluster scheduler enforces configured execution policy.

Computation clusters that are the topic of interest in this paper are usually connected to the fast storage system which is shared among all the computing nodes. Access to this storage may present a bottleneck for some applications in the system and should be simulated for such workload types.

III. DISCRETE EVENT SIMULATION

There are three main approaches to discrete event simulation: using simulation tools, using simulation libraries and creating a custom simulation.

The use of general purpose simulation tools is a most abstract way to perform the system simulation. It limits the number of available features, but hides away the complex execution details. The limited number of elements in the simulation can make model creation difficult if complex behaviour needs to be modelled.

Simulation libraries provide access to basic simulation operations, and enable the programmer to design the custom model using the given platform. This grants more modelling freedom, but some aspects of the model may not be covered. For instance, event cancellation is not implemented in some of the simulation libraries, but may be required naturally by the model. There is a workaround that can be made, but this often leads to more complicated model.

Programming a custom simulation gives all the power and flexibility to the modeller, but is most time consuming, and requires detailed insight into discrete event simulation principles.

Since our goal is the design of the concurrent simulation system for comparison of scheduling algorithms, we decided to create the system from scratch using insights to existing simulation libraries. It is our objective to provide an extensible simulation framework customized for the modelling of cluster scheduling algorithms.

Several executives [7] for discrete event simulation exist, with process based executive and three-phase executive being the most efficient. Executive is an algorithm at the core of the simulation that tracks events in the model and executes appropriate actions.

Process based executive is most commonly used, because it allows the greatest flexibility for the modeller and has a steep learning curve. Elements of the model using process based executive are designed as processes with conditional and unconditional delays. In the simulation each of the processes is executed until unconditional or conditional delay occurs.

Three-phase simulation creates a distinction between unconditional and conditional delays, and imposes an order on their execution. Unconditional delays and associated actions are called B activities and are executed before conditional events and their respective actions at any time. Conditional delays and actions are called C activities.

The three-phase executive is composed of three phases. At the A phase the time for the first B activity occurrence is determined. All the B activities for that time are executed during the B phase. B phase is followed by the C phase at which all the C activities are performed. C activities are repeatedly scanned until no activity can be executed because execution of any C type activity can change the state of simulated system and this can cause other C activities eligible for execution. The modelled order of C activities enables designer to avoid ambiguities in the model.

Three-phase simulation approach requires better understanding and detailed dissection of the model that require additional modelling time. The strict ordering of conditional activities gives greater control and helps avoiding potential deadlocks and race conditions in the model when compared to the process based executive. Deadlocks and race conditions are also avoidable using process based approach, but the analysis of their potential occurrence is complicated due to an inherent lesser transparency of the model.

IV. Simulation for Clusters and Grids

Cluster simulations are usually contained within the more complex grid simulators. Most of them like Optorsim [8], SimGrid [9], GridSim[10], Bricks[11] are focused on data replication and process migration among clusters and detailed support for scheduling policies within the cluster is not provided.

SimGrid is aimed at the simulation of grids on real hardware in order to capture fine interactions among parallel applications and IO storage operations.

The most widely used Gridsim tool that is still being maintained employs process based simulation executive simjava [12] and defines an extensible cluster allocation policy. Sample space shared and time shared policies are provided to the developers.

Beosim [13] tool handles cluster simulation to a greater detail, but the source code and the inside specifics are not publicly available.

Scheduling simulations for clusters are mostly done manually by the developers, and are specific for the scheduling policy and the simulated system. This makes unbiased comparison of scheduling strategies harder for researchers since experimenting in different environments and lack of implementation specifics for different scheduling policies may lead to a different resulting performance.

V. Three-Phase Cluster Scheduler

Three-phase cluster scheduler is proposed in order to support investigation of the scheduling algorithm performance. Requirements imposed on the scheduler, architecture of the entire system and some of the simulation details are presented in the following subsections.

A. Requirements

Requirements are a prerequisite for any system design and modeling process. In this paper we target both space shared and time shared systems, and would even like to support simulation of hybrid systems that allow time sharing on subset of computing resources, while enforcing space sharing on the rest of the system. This may prove beneficial for interactive applications where instant response is more important than overall performance of the target application.

Detailed modeling of data flow within the large computer cluster is not feasible, but data staging within the cluster may be interesting to simulate since data storage can be identified as a bottleneck in certain applications.

Inputs to the simulation system are job sequences, scheduling policies and metrics of interest. Job sequence can be either statistically modeled or a real job trace [14]. Statistically modeled sequences are more general and provide unlimited number of inputs, while the real job traces are bounded, but provide an insight to the real world system behavior.

The simulation system must be easily extendable in order to enable design, comparison and refinement of different scheduling policies. Different metrics and data of interest should be easily exposed and analyzed within the system. Model development and refinement cycle must be shortened to enable many different algorithms and parameter sets to be evaluated. This can be done by the real time performance monitoring of the simulation, and by visualization of the different output parameters.

B. System architecture

The simulation system is composed of several parallel discrete event simulations that enable real time comparison of different scheduling strategies as presented on Fig 1. *Simulation designer* is a user controllable component that

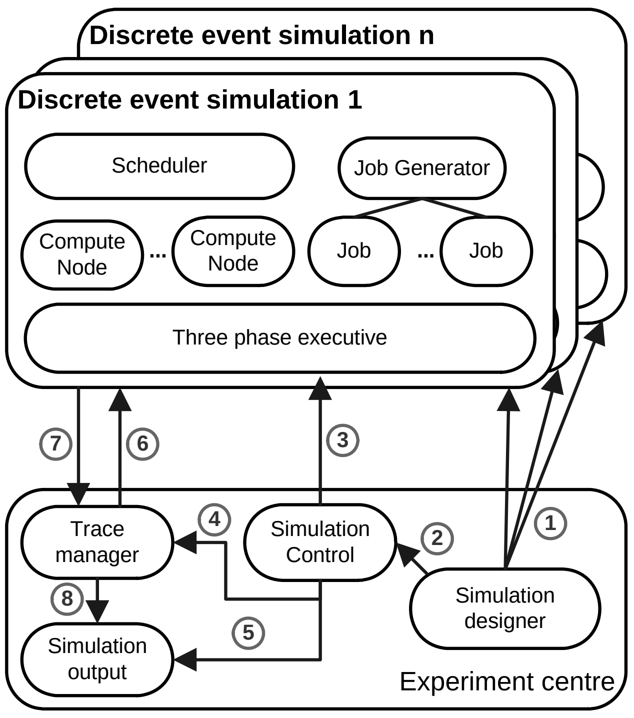


Fig. 1. Simulation system architecture

is used to define (1) parameters of every simulation in the experiment. Parameters include type and settings of the scheduling policy, job generator settings and compute node servicing policy.

When simulation is configured (2), *Simulation control* component is used to specify parameters and points of interest for the simulation (4) that will be collected and to define output of the simulation (5) in terms of these parameters. *Trace manager* component subscribes (6) to the events of interest and passes the data gathered from the simulation to the *Simulation Output* component that visualizes simulation results (7).

Simulation control component is used to control the execution of the discrete event simulations (3). It directs the execution of the phases at the three-phase simulation executive. At the end of each A phase, every discrete simulator controlled by the component announces its next event time. By using that information and by selectively choosing phases from different executives, *Simulation Control* component synchronizes clocks in different discrete simulations providing continuous comparison of parameters at the *Simulation Output* component.

Features like asynchronous running, pausing and stepping through different simulations can be easily implemented in this architecture by providing the user interface and changing execution rules at the *Simulation Control*.

C. Three-phase simulation entities

Identification and classification of entities and resources is one of the most important issues in system simulation. For each element participating in the simulation a choice is made by determining its role and the simulation goal.

Basic elements in the scheduling simulation are jobs, computing nodes and the scheduler. Jobs are accompanied with the respective job generator that creates new jobs according to the specified policy. Since all the basic elements are unique they are represented as entities in the simulation.

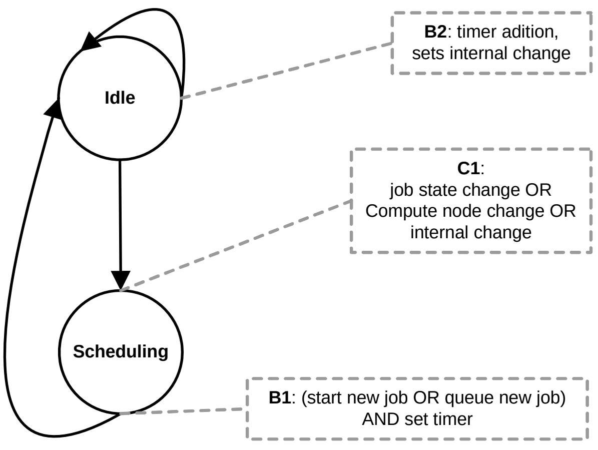
Scheduler is the central point of the simulation and is modeled as an active system entity. It is responsible for data staging and scheduling jobs to computing nodes which includes handling of the hardware malfunction or regular maintenance.

Scheduler element (Fig. 2) constantly switches between *idle* and *scheduling* state. In order to enter the *scheduling* state, a change in the system state that requires scheduling action must occur (*condition C1*). There are three system state changes that trigger the scheduling process: a job state change, a compute node state change and an internal change.

Typically, scheduling is considered to be an almost instant activity in the majority of the cluster systems. This is required because the rate of the scheduling decisions should closely match the job arrival rate. Some heuristic algorithms such as genetic algorithms [15] are slower and their reaction to system changes must be modeled with a delay. It is assumed that scheduler decision is to be made at the *B1* activity using the information that was available in the system at the time of the *C1* activity. In order to spare the computation resources the decision is made as part of the *C1* activity and the action that is a consequence of that decision is taken during the *B1* activity. Duration of the scheduling state depends on the type of algorithm used and the amount of input data in the system that are currently available. State *scheduling* and *B1* activity can be omitted when fast scheduling policies are modeled such as first-come-first-served (FCFS) or different types of backfilling.

When time shared scheduling algorithms are simulated, the interval for the algorithm time slice is setup at the *B1* activity. Handling of the time slice interrupt, at which context switch of the running processes is done, is performed while algorithm is in the *idle* state. Execution of the *B2* activity changes internal system state, which effectively triggers the scheduler to switch from *idle* to the *scheduling* state.

The only issue with this approach is that potentially existing *B2* event must be canceled at the *C1* activity. This is due to inability of the three-phase scheduling approach



Idle
Scheduling
B2: timer adition, sets internal change
C1: job state change OR Compute node change OR internal change
B1: (start new job OR queue new job) AND set timer

Fig. 2. Scheduler activity diagram

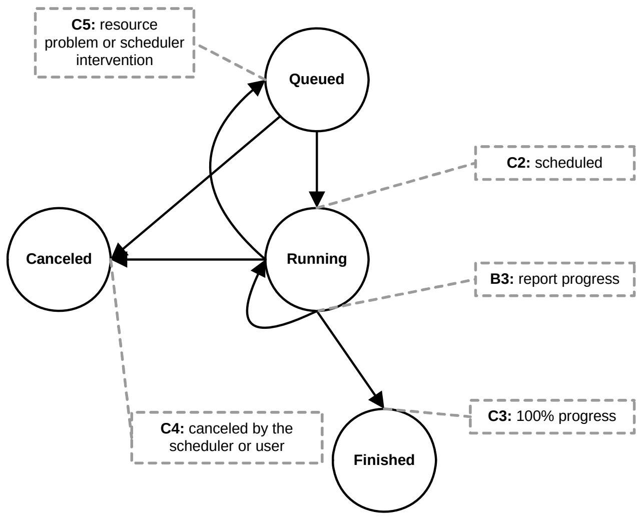

C5: resource problem or scheduler intervention
Queued
C2: scheduled
Canceled
Running
B3: report progress
C3: 100% progress
C4: canceled by the scheduler or user
Finished

Fig. 3. Job activity diagram

to handle more than one *B* type event pending at any moment which may happen when one *B2* is pending and change in the system starts the execution of *C1*. Since execution of *C1* sometimes involves generation of *B1* events that are handled in the future, other *B* type events are not allowed. This doesn't change semantics of the time sharing algorithm since new *B2* event is scheduled at the *B1* activity, and it is not expected for processes in the time sharing system to run while being rescheduled or context switched.

Event canceling must be supported in the underlying three phase discrete event simulation package in order to terminate pending *B2* event at C1 activity. For simulation of space shared scheduling algorithms, *B2* activity is not used, and there is no need for the event cancellation.

Semantics of the internal change vary for different scheduling policies and can include data staging notifications or different grid requests. In order for some internal changes to occur, external entity must be created that affects the scheduler internal state. Scheduler should also be extended to handle the new functionality.

Jobs in the system that are created using job generator are represented with an entity in the simulation. Job activity state diagram is given in the Fig 3. Jobs are initially queued and the event of a job creation triggers scheduler to perform its decision cycle. During the decision cycle some of the jobs can be scheduled for execution, and appropriate attributes are set for both jobs and the target computing nodes. When the job is marked for the execution, *C2* activity is performed which calculates duration of the job on the given hardware and sets the job to the *running* state. Running jobs can occasionally report its progress to the scheduler through the *B3* activity. Jobs that are not designed to report its progress perform *B3* only on completion.

Jobs are allowed to be stopped for a number of reasons that are matched with the activities in the simulation. Running job can be interrupted because of the outages of some of the computing nodes or because scheduler decided to stop it. This is handled by the activity *C5* which puts job in the *queued* state and changes some attributes of the job depending on the type of the job. Preemptive jobs can be moved to other resources on the cluster and their current progress and collected statistics must be saved. Other jobs need to be restarted.

Jobs that are canceled by the user or by the scheduler and are not supposed to continue are handled by the activity *C4*. Jobs that are either in *queued* or *running* state can be canceled. The activity *C4* also notifies all the computing nodes acquired by the job that the execution is discontinued.

When the job is completed *C3* activity is performed that moves the job to the *finished* state and updates statistics for that job. The computing nodes are also informed about the job completion.

Computing nodes in the system are modeled using activity diagram in Fig. 4. Every computing node can be in *idle*, *running* and *out_of_service* state. Initiatially all the nodes are in the *idle* state, and there is one *B4* event scheduled.

When scheduler assigns the job to the computing node the change in the job state triggers the activity *C6*, which causes computing node to change state from *idle* to *running*. When the job is finished or stopped it changes its internal state, which causes activity *C7* to move computing node from *running* to *idle* state.

From *idle* and *running* states the computing node moves to *out_of_service* state when event *B4* occurs. New *B5* event is scheduled as a part of the *B4* activity. When servicing of the cluster component is finished, *B5* activity is executed. This activity changes the state of the computing node to *idle* and schedules new *B4* event.

The state change of the computing node causes scheduler to perform its scheduling cycle.

D. Implementation

Object oriented languages are a perfect match [16] for the three-phase simulation algorithm. This distributed approach, where executive runs on a server and simulation components reside on the client, is beneficial in general, although slightly modified version that drops the distributed approach is more appropriate for custom simulations. Keeping the simulation set within the application boundaries avoids potential performance penalty in the distributed environment, but enforces the users to run and maintain the simulation locally.

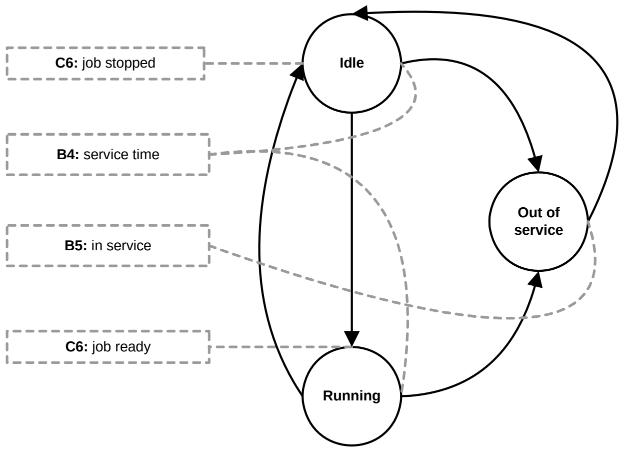


Fig. 4. Computing node activity diagram

There are two threads of execution that handle both user interface and the three-phase simulation execution. Threads communicate in two ways: user interface thread controls the execution of the simulation, while simulation thread reports data changes to the user interface in order to enable visualization of current simulation results.

At heart of the simulation thread is an *Executive* class that performs three phases of the discrete event simulation. The rest of the simulation is encapsulated in the *Simulation* class that instantiates the entities and sets up the simulation.

System components in object oriented simulation are represented by classes, and B activities are usually handled by methods within these classes. This is valid since entities are required to wait for B events and they typically change their internal state when time is due.

There is an issue with C activities because their conditional execution typically depends on several different entities in the system so implementing them within an entity class would conflict the encapsulation property of the object oriented paradigm. On the other hand, some of the C activities are conditionally bound to the rest of the system but their effect is only visible to their respective entity so it would make sense to implement such activities within the appropriate entity class. It is decided to classify C activities according to their dependencies on other entities, and to put the ones that affect only one entity type within this entity's class, and to implement the others within the *Simulation* class. This would partly violate encapsulation principle, but would lesser the clutter in the *Simulation* class.

In order for simulation to publish its results, each entity in the system must expose data to subscribers, and raise events in case the subscribed data changes its value. This behavior is expected to slow the simulation process, but real time performance comparison of the algorithms allows earlier debugging of the model.

VI. CONCLUSION

In this paper we presented the basic building blocks and the architecture of the cluster simulator. Three phase simulation technique is selected as the basis for the simulation due to its inherent lack of ambiguity. Essential simulation entities are recognized and modeled to a detail capturing important aspects of cluster schedulers as well as variety of scheduling approaches. Implementation pointers for the object oriented environments are also given.

Performance of the simulator is important when large systems are simulated. Three phase simulation approach should perform faster then process based simulations if properly implemented because of the context switching that is unavoidable in the latter.

In order to increase simulation performance several modifications can be made. Simulation of the entire system can be distributed to many machines resulting in faster simulation. Event based runtime notifications to the modeler can be avoided and the simulation trace can be used to analyze system behavior.

The proposed simulator should be extended to support basic grid operations and to allow communication between clusters. This would require modification of the scheduler entity, and the simulation of the grid interconnection network.

REFERENCES

[1] TOP500 Supercomputing Sites, http://top500.org/

[2] D. G. Feitelson, “Metrics for parallel job scheduling and their convergence”, Lecture Notes in Computer Science, vol. 2221/-1/2001, p. 188-205, 2001

[3] D. Bernstein, M. Rodeh and I. Gertner, “On the Complexity of Scheduling Problems for Parallel/Pipelined Machines“, IEEE Transactions on Computers, vol. 38, p. 1308, 1998.

[4] D. Jakobović, L. Jelenković and L. Budin, “Genetic Programming Heuristics for Multiple Machine Scheduling“, Lecture Notes in Computer Science, vol. 4445, p. 321-330, 2007.

[5] D. Jakobović and L. Budin, “Dynamic Scheduling with Genetic Programming“, Lecture Notes in Computer Science, vol. 3905, p. 73-84, 2006.

[6] Condor Project Homepage, http://www.cs.wisc.edu/condor/

[7] M. Pidd, *Computer Simulation in Management Science*, 5th Edn., John Wiley & Sons, West Susex, England, 2004

[8] W. H . Bell, D. G . Cameron, L. Capozza, A.P. Millar et al., “OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies“, International Journal of High Performance Computing Applications, vol. 17, p. 2003, 2003.

[9] H. Casanova, “SimGrid: A toolkit for the simulation of application scheduling“, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, p. 430-437, 2001

[10] R. Buyya and M. Murshed, “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing “, The Journal of Concurrency and Computation: Practice and Experience (CCPE), vol. 14, 2002.

[11] A. Takefusa, H. Casanova, S. Matsuoka and F. Berman, “A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid“, Proc. of 10th IEEE International Symposium on High Performance Distributed Computing, p. 406-415, 2001

[12] F. Howell and R. Mcnab, “simjava: A discrete event simulation library for Java“, In International Conference on Web-Based Modeling and Simulation, p. 51-56, 1998

[13] BeoSim - Multicluster Computational Grid Simulator for Parallel Job Scheduling Research http://www.parl.clemson.edu/~wjones/research/

[14] D.G. Feitelson, “Workload modeling for performance evaluation”, Lecture Notes in Computer Science, vol. 2459, p. 114-141, 2002.

[15] A.J. Page and T.J. Naughton, “Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing”, In the Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing, p. 189., 2005.

[16] M. Pidd and R. A. Cassel, ” Three phase simulation in Java”, In Proceedings of the 1998 Winter Simulation Conference, p. 367-371, 1998.