Divisible Load Scheduling for Grid Computing

Dantong Yu* and Thomas G. Robertazzi**
*Department of Physics
Brookhaven National Laboratory
Upton, NY 11973, USA
**Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY 11794, USA
dtyu@bnl.gov and tom@ece.sunysb.edu

Abstract

The use of divisible load scheduling theory is proposed to model and design grid systems such as those arising in large physics experiments. Current divisible load theory is summarized. A typical application, the STAR experiment at RHIC is discussed. This includes a sample calculation based on existing infrastructure numbers.
Keywords: divisible load scheduling theory(DLT), grid computing.

1 Introduction

Scheduling, performance prediction and resource management are important but challenging tasks for grid computing efforts. In particular, there has been existing and ongoing work on network batch queueing and wide area scheduling. Ian Foster and others have written [16, 17] on the nature of the grid scheduling environment with its multiple possible performance measures and constraints, its large number of jobs and job sources and its large number of processing, transmission and storage resources.

Efforts in grid technology would benefit from a robust mathematical, but tractable performance prediction and scheduling tool. A natural first choice, queueing theory, is tractable for simpler models but has tractability limitations as queueing statistics become more complex (i.e. realistic). This is particularly true of large systems. Moreover the statistical assumptions often made may only be approximate, or worse, in reality.

In this paper we propose the use of divisible load (data) scheduling theory for grid scheduling and performance prediction. Based on divisible (partitionable) loads, the resulting continuous variable theory is a linear one with all of the attendant tractability that implies. The theory is also fundamentally deterministic, lacking the weakness of statistical assumptions. One can also, as a number of researchers have done, model a large number of small tasks as a divisible load [15].

The next section presents an overview of divisible load scheduling research. This is followed by a discussion of a representative example, the STAR experiment in the RHIC project. A simple divisible

calculation using typical grid infrastructure numbers is presented to illustrate the suitability of divisible load theory for grid problems, followed by the paper's conclusion.

2 Divisible Load Scheduling Theory

Over the past fifteen years or so, a new mathematical tool has been created to allow tractable performance analysis of systems incorporating communication and computation issues, as in parallel and distributed processing. A key feature of this divisible load distribution scheduling theory (known as **DLT**) is that it uses a linear mathematical model. Thus as in other linear models such a Markovian queuing theory or electric circuit theory, divisible load scheduling theory is rich in such features as easy computation, a schematic language, equivalent network element modeling, results for infinite sized networks and numerous applications. This linear theory formulation opens up striking modeling possibilities, including for grid computing.

Since the original work on this subject in 1988, by one of the co-authors and James Cheng [2], then a graduate student, an expanding body of work on this subject has been published on a worldwide basis [1, 12]. The original motivation on the part of the P.I. for this 1988 work was intelligent sensor networks with elements which could measure, communicate and compute. The divisible load concept has subsequently been applied to parallel and distributed computing.

In divisible load distribution theory it is assumed that computation and communication loads can be partitioned arbitrarily among a number of processors and links, respectively. A continuous mathematical formulation is used. In addition, there are no strong precedence relations among the data. Therefore load can be arbitrarily assigned to links and processors in a network. This class of problems is well suited for modeling a large class of data parallel computational problems. As a secondary benefit, it sheds light on architectural issues related to parallel and distributed computing. Moreover, the theory of divisible load scheduling is fundamentally deterministic. While stochastic features can be incorporated, the basic model has no statistical assumptions that can be the Achilles' heel of a performance evaluation model.

A typical divisible load model allows one to account for model parameters such as heterogeneous processor and link speed and computation and communication intensity. For instance, the time to process the load at the ith processor is typically $\alpha_i \omega_i T_{cp}$ where α_i is the fraction of the total load assigned to the ith processor, ω_i is the inverse speed of the ith processor and T_{cp} is the computation intensity of the load. Similarly, the time to transmit a load fraction α_i on the ith link is $\alpha_i z_i T_{cm}$ where z_i is the inverse speed of the ith link and T_{cm} is the communication intensity of the load.

Divisible load models can typically be solved algebraically for optimal allocation and timing of load to processors and links. Here optimality is defined in the context of the specific interconnection topology and scheduling policy used. A solution is usually obtained by forcing all processors to finish computing at the same time (otherwise, intuitively, load could be transferred from busy to idle processors see [5] for a formal

proof).

In the case of a single level tree (star) network this constraint leads to a series of chained equations that can be solved recursively for the optimal allocation of load to each processor in the context of a given load distribution schedule. The recursions are similar in form, though not parameters, to the recursive solutions of equilibrium state probability for a single Markovian queues [5, 6]. In the worst case, for a more general network such as a multilevel tree network, by forcing all processors to stop computing at the same time one can set up a set of N linear equations for N processors [3]. These equations can be solved for the optimal allocation of load to processors through standard linear equation solution techniques.

A useful concept in divisible load theory is that of an equivalent processor [4]. One can replace a sub-network of a larger network with a single equivalent processor that has exactly the same overall computing power (i.e. speed) as the original sub-network. This equivalent processor concept is analogous to the equivalent element of electric circuit theory and the equivalent queue of queueing theory. Moreover for homogeneous networks, by setting up an implicit equation, it is possible to solve for the performance of infinite size networks.

Also important is the actual load scheduling policy used. Linear divisible load theory can model a wide variety of approaches. For instance, one can distribute load either sequentially or concurrently. Under sequential (bus like) load distribution, the policy used in most of the literature to date, a node will distribute load to one of its children at a time. This results in a saturating speedup as network size is increased. One could improve performance by distributing load from a node to children in periodic installments but performance still saturates as the number of installments is increased [1]. A superior performance results if load is distributed concurrently. That is a node distributes load simultaneously to all of its children. In 2002 it was shown that such concurrent load distribution is scalable for a single level tree in the number of children nodes (i.e. linear growth in speedup as the number of children nodes increases) [13]. Here speedup is the ratio of solution time on one processor to solution time on an N processor network. Speedup is a measure of parallel processing advantage.

Other scheduling policy features that can be modeled are store and forward and virtual cut through switching and the presence or absence of front end processors. Front end processors allow a processor to both communicate and compute simultaneously by assuming communication duties. Also, the time to report solutions from processors back to a load originating processor can be modeled.

A number of model features studied for divisible scheduling include finite memory [9], fixed start up charge models [7], processor release times and load granularity. Divisible load scheduling theory has been applied to a variety of applied problems including monetary cost optimization, matrix calculations [8], multimedia scheduling and image processing [11]. There has also been experimental work [10, 11] and a proposal for dynamic real time scheduling [14].

3 Suitability of Divisible Load Theory For Grid Problems

High energy and nuclear physics experiments requires effective analysis of large amounts of data by widely distributed researchers who must work closely together. Expanding collaborations (envisioned by the Particle Physics Data Grid [21]) and demanding data analysis requirements coupled to increasing computational and networking capabilities are giving rise to a new class of network-based computing platforms: globus [16], condor [19], LSF [20], and so on. All of these recently emerging platforms require a scheduling strategy to efficiently make use of distributed computers, high-speed networks and storage resources.

3.1 STAR Computing Needs

The Solenoidal Tracker At RHIC (STAR) collaboration is a large international collaboration of about 400 high energy and nuclear physicists located at 40 institutions in the United States, France, Russia, Germany, Israel, Poland, and so on. After the Relativistic Heavy-Ion Collider at Brookhaven National Laboratory came on-line in 1999, STAR began data taking and concurrent data analysis that will last about ten years. STAR needs to perform data acquisition and analyzes over approximately 250 tera-bytes of raw data, 1 peta-bytes of derived and reconstructed data per year. The sequence of data flow is:

- *Raw Data Generation*: The STAR detector writes raw data to tapes stored in tape silos.
- *Reconstruction*: The RAW data will be processed individually for event reconstruction. The reconstructed events are written to the storage system.
- *DST generation*: The reconstructed events will be processed, the identified particles suitable for physics analysis will be generated from the reconstructed events by calibrating the tracking of events. The results of this process is called DST (Data Summary Tape).
- *Micro-DST generation*: For different physics analysis, a subset of events from the DST is selected with the corresponding filters on the type of events. The subset is called Micro-DST.

Physicists submit jobs to the data grid. Each job generally consists of many subjobs. Some large jobs might include thousands of subjobs. Each subjob has a run time of seconds up to days. STAR subjobs do not communicate with each other directly. A subjob has one or more file sets as its input, creates new output files or updates existing files. All of these processes are applied to individual datasets. There is no co-relation between the processes on two different datasets.

3.2 Hardware Description

Two main computing resources of similar size, RCF (BNL) and PDSF/NERSC (LBNL) STAR have 1000 CPUs, 30 TB disk, and HPSS at both sites. Production-level processing is done at both sites independently: DST production at BNL, embedding at LBNL There is a simple model for user-level computing: Almost everything is available at both sites on disk. The STAR experiments use regular UNIX files databases to

store experimental results. Typical database file sizes are approximately 1 gigabytes. The size of the total dataset is 50TB at this moment. When a user submits job requests, this job will typically be broken down to several hundred small jobs; each job will work on a 1 gigabyte file. Some (not all) STAR/RHIC computing resources are shown in Table 1. The STAR storage is distributed among five NFS servers, each of them has a one gigabit network connection.

Number of Hosts	Vendor	CPU Speed (MHz)	Number of CPUs Per Host	Memory (MB)	Network Bandwidth (mbps)	Host ID
135	VA Linux	450	2	512	100	1
161	VA Linux	800	2	512	100	2
301	IBM	1000	2	1024	100	3
155	IBM	1400	2	1024	100	4

Table 1: STAR computing resource chart.

3.3 Distributed Job Scheduling

Optimizing job execution time by efficiently scheduling subjobs to different sites and individual computers is crucial to successful physics research in the grid computing environment, due to the great computing need for physics processes and the limited computing resources. Many approaches have been proposed to move the physics programs to the data repository; to schedule the jobs/subjobs to the least loaded computing nodes or the fastest computing node first; or to pre-schedule data strategically for max reuse and schedule jobs thereafter. But most of these approaches do not reflect the physics reality: large amounts of jobs which are divisible with no interprocess communication between jobs and subjobs. Many algorithms can not schedule jobs efficiently in either of two ways: the computing sites without data are idle while the computing sites with the needed data are overwhelmed because of unbalanced load sharing; or the time is spent on transferring data while the most computing resource are waiting for data transfer. We propose balanced load sharing through the use of divisible load distribution scheduling theory and algorithms which are targeted towards the characteristics of physics processes, and provide the best effort scheduling on computing and data resource to avoid “hot spots” in the grid.

3.3.1 STAR Simulation

Large numbers of simulated events are necessary for calculating the efficiency of the STAR detector, optimizing the STAR detector and planning the future data analysis. Simulation processes mimic the particle production from nucleus + nucleus collisions, thereafter the propagation, scattering of the particles throughout the detector and the response of each sub-detector to the passage of particles [18]. Simulation generates simulated events. Here we consider a STAR physics process which simulates gold + gold collision. The input to this physics process is a set of parameters for each event generator; and the output is 35160 generated events. The event are stored in 100 files, which each files is about 1.352 GB. It takes about 24 hours to generate 1000 events on a host with dual 1GHz CPUs.

3.3.2 Event Reconstruction

The STAR event reconstruction software converts the raw input data from either simulations or from the real experiment into a reconstruction of the particle production from gold + gold collisions. The output of the event reconstruction software is a list of particles corresponding to the input event and includes the momentum, energy charge and other physics characteristics [18]. The average size of an input raw event is 3.84 MB; and the size of an output reconstructed event is 1.527MB. It takes an average of 652.39 seconds to reconstruct a single event. Table 2 lists the symbols and their definitions in this example.

Symbols	Definitions and Values
α_i	fraction of load assigned to the ith host
ω_i	inverse host i speed
z_i	inverse network link i speed to the ith host
T_{cp}	computing intensity constant of processing a unit load on the standard host
T_{cm}	communication intensity constant of transferring a unit input/output load on a standard link
T_{cm_input}	communication intensity constant of transferring a unit input load on a standard link
T_{cm_output}	communication intensity constant of transferring a unit output load on a standard link
$\alpha_i\omega_i T_{cp}$	time to process the ith fraction on the ith host
$\alpha_i z_i T_{cm}$	the time to transmit the fraction α_i of the entire load over the ith link
ρ	the ratio between communication and processing intensities, $\rho = \frac{T_{cm}}{T_{cp}}$

Table 2: Symbol Definitions and Parameter Values for the experiments

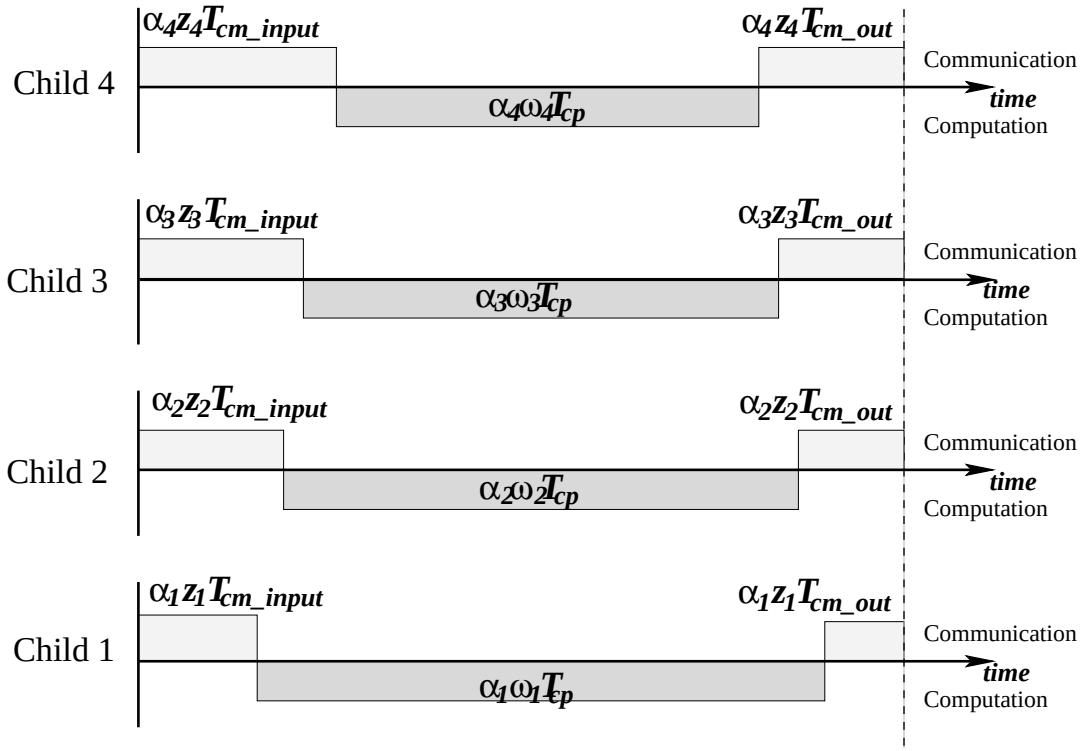

Figure 1: Timing diagram of star tree with simultaneous distribution; each child pulls/push data from/to storage system.

We assume that the hosts are exclusively used for event reconstruction. A STAR job normally generates 500 events during each run. This job is scheduled to run on four hosts with different CPU speed in this example. The specification of the four hosts are shown in Table 1. We choose a computer with 1 Hz CPUs as the standard host. From Table 1, we obtain $\omega_1 = 1.11 \times 10^{-9}$, $\omega_2 = 6.25 \times 10^{-10}$, $\omega_3 = 5 \times 10^{-10}$, and $\omega_4 = 3.5714 \times 10^{-10}$. In this example, we choose 500 events as a unit load. Here T_{cp} is the computing intensity of generating these 500 events on the standard host. To calculate the value of T_{cp}, we use the reference figures in the paragraph above and Table 1, and ω_3 is the inverse speed of a dual 1Ghz host. We already know $\omega_3 T_{cp} = 500 \times 652.39 = 326195$ seconds, therefore $T_{cp} = 6.52390 \times 10^{14}$. The subjobs will not start until the required events are available. The bandwidth of the network backbone is much bigger than host network bandwidth. Thus the communication speed is only determined by the host network bandwidth. All of these hosts in Table 1 use fast Ethernet. Thus $z_1 = z_2 = z_3 = z_4 = 10^{-8}$. Naturally if a network is used as a shared resource by a large number of hosts, the following equations still hold if we approximate transmission speed by its effective value. More comprehensive discussion regarding the communication cost can be found in [15]. The timing diagram in Figure 1 shows the input/output communication delay above the time axis and computing time below the time axis. Since the network speed is constant, we combine T_{cm_input} and T_{cm_output} as T_{cm}, while $T_{cm} = T_{cm\_input} + T_{cm\_output}$. In this example, $z_1 T_{cm}$ is the communication intensity of transferring a unit load over network link 1, $z_1 T_{cm} = 500 \times \frac{(3.84+1.527)MB}{100mbps} = 500 \times \frac{42.936mb}{100mbps} = 214.68$ seconds. Therefore $T_{cm} = 2.1468 \times 10^{10}$. The finish times for all hosts should be equal for optimal load distribution [5]. The following equations [13] hold:

$$
\begin{aligned}
\alpha_1\omega_1 T_{cp} + \alpha_1 z_1 T_{cm} &= \alpha_2\omega_2 T_{cp} + \alpha_2 z_2 T_{cm} \\
\alpha_2\omega_2 T_{cp} + \alpha_2 z_2 T_{cm} &= \alpha_3\omega_3 T_{cp} + \alpha_3 z_3 T_{cm} \\
\alpha_3\omega_3 T_{cp} + \alpha_3 z_3 T_{cm} &= \alpha_4\omega_4 T_{cp} + \alpha_4 z_4 T_{cm} \\
\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 &= 1,
\end{aligned}
$$

$$
\begin{aligned}
\alpha_2 &= \frac{\omega_1 T_{cp} + z_1 T_{cm}}{\omega_2 T_{cp} + z_2 T_{cm}}\alpha_1 = f_1\alpha_1 = f_1\alpha_1 \\
\alpha_3 &= \frac{\omega_2 T_{cp} + z_2 T_{cm}}{\omega_3 T_{cp} + z_3 T_{cm}}\alpha_2 = f_2\alpha_2 = f_2 f_1\alpha_1 \\
\alpha_4 &= \frac{\omega_3 T_{cp} + z_3 T_{cm}}{\omega_4 T_{cp} + z_4 T_{cm}}\alpha_3 = f_3\alpha_3 = f_3 f_2 f_1\alpha_1 \\
\alpha_1 &= \frac{1}{1 + f_1 + f_2 f_1 + f_3 f_2 f_1}.
\end{aligned}
$$

Based on these equations, $\alpha_1 = 12.34\%$, $\alpha_2 = 21.92\%$, $\alpha_3 = 27.39\%$, $\alpha_4 = 38.34\%$, and the job processing time is 89419.4 seconds. Note that solution time is $T_{solution} = \alpha_i(z_i T_{cm} + w_i T_{cp}), \forall i = 1, 2, 3, 4$. A processor with inverse processing speed equivalent to the entire network is: $\omega_{eq} = \frac{z_1\rho+\omega_1}{1+f_1+f_2 f_1+f_3 f_2 f_1}$. For a homogeneous network, ω is the inverse host speed, $speedup = \frac{\omega T_{cp}}{\omega_{eq} T_{cp}} = (\frac{\omega}{z_1\rho+\omega})(1 + f_1 + f_2 f_1 + f_3 f_2 f_1)$. Naturally for a homogeneous network, the f_i's are identical.

It can be seen that the STAR example is much more computation intensive than communication intensive. The type of modeling is easily modified to account for any number of processors and different scheduling policies and interconnection topologies.

4 Conclusion And Future Works

Grid technology will benefit from a tractable and flexible analytic design tool. Divisible load scheduling theory is a promising candidate for this role. Open challenges include the further development of the theory to model very large grid systems and their associated problems. We will implement the divisible load scheduling algorithm, install it on a grid type computing facility, compare its performance with other local batch systems, such as LSF, condor, and PBS.

5 Acknowledgments

We especially thank Jérôme Lauret for explaining to us the complex STAR physics concepts and providing real life STAR job parameters for our examples. We also thank Lidia Didenko and Maxim Potekhin for explaining the STAR simulation processes and for valuable discussions. Thomas Robertazzi's work is supported by NSF grant CCR-99-12331. Dantong Yu's work is supported by DOE PPDG/ATLAS/RHIC grants.

References

[1] Bharadwaj, V., Ghose, D., Mani, V. and Robertazzi, T.G., *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos CA, Sept. 1996, 292 pages.

[2] Cheng, Y.C. and Robertazzi, T.G., "Distributed Computation with Communication Delays", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 24, No. 6, Nov. 1988, pp. 700-712.

[3] Cheng, Y.C. and Robertazzi, T.G.,"Distributed Computation for a Tree Network with Communication Delays", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 26, No. 3, May 1990, pp. 511-516.

[4] Robertazzi, T.G., "Processor Equivalence for a Linear Daisy Chain of Load Sharing Processors", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 29, No. 4, Oct. 1993, pp. 1216-1221.

[5] Sohn, J. and Robertazzi, T.G., "Optimal Load Sharing for a Divisible Job on a Bus Network", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 32, No. 1, Jan. 1996, pp. 34-40.

[6] Bataineh, S., Hsiung, T., and Robertazzi, T.G., "Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job", *IEEE Transactions on Computers*, Vol. 43, No. 10, Oct. 1994, pp. 1184-1196.

[7] Blazewicz, J. and Drozdowski, M., "Distributed Processing of Divisible Jobs with Communication Startup Costs", *Discrete Applied Mathematics*, Vol. 76, Issue 1-3, June 13, 1997, pp. 21-41.

[8] Ghose, D. and Kim, H.J., "Load Partitioning and Trade-Off Study for Large Matrix Vector Computations in Multicast Bus Networks with Communication Delays", *Journal of Parallel and Distributed Computing*, vol. 54, 1998.

[9] Li, X., Bharadwaj, V. and Ko, C.C., "Divisible Load Scheduling on Single Level Tree Networks with Buffer Constraints", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 4, Oct. 2000, pp. 1298-1308.

[10] Drozdowski, M. and Wolniewicz, "Experiments with Scheduling Divisible Tasks in Clusters of Workstations", in in Bode, A., Ludwig, T., Karl, W. and Wismueler, R., editors, EURO-Par-2000, *Lecture Notes in Computer Science no. 1900*, Springer-Verlag, 2000, pp. 311-319.

[11] Bharadwaj V. and Ranganath, S., "Theoretical and Experimental Study of Large Size Image Processing Applications using Divisible Load on Distributed Bus Networks", *Image and Vision Computing*, Elsevier Publishers, Sept. 2002.

[12] Bharadwaj, V., Ghose, D., Robertazzi, T.G., "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems", in Ghose, D. and Robertazzi, editors, special issue of *Cluster Computing* on Divisible Load Scheduling, Jan. 2003, Vol. 6, No. 1, pp. 7-18.

[13] Hung, J.T. and Robertazzi, T.G., "Scalable Scheduling in Parallel Processors", *Proceedings of the 2002 Conference on Information Sciences and Systems*, Princeton University, Princeton NJ, March 2002 (six pages).

[14] Ghose, D., "A Feedback Strategy for Load Allocation in Workstation Clusters with Unknown Network Resource Capabilities using the DLT Paradigm", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, Nevada, USA June 2002, Vol. 1, pp. 425-428.

[15] Beaumont, O., Carter, L., Ferrante, J., Legrand A. and Robert, Y., "Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms", *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)*, June 2002.

[16] Foster, I. and Kesselman, C., eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufman, 1999.

[17] Ranganathan, K. and Foster, I., "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications", *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.

[18] Bossingham, R., Christie, W., LeCompte, T., Lisa, M., Llope, W.J., Margetis, S., Pruneau, C., Ray, L., Sakrejda, I., Wilson, W.K., and Yepes, P., "STAR Offline Simulations and Analysis Software Design", *STAR Note 0281*, http://www.star.bnl.gov/STAR/html/sas_ldesignsas_design_version1_0.ps, January 1997.

[19] Litzkow, M., Livny, M., and Mutka, M. W., "Condor - A Hunter of Idle Workstations", *Proceedings of the 8th International Conference of Distributed Computing Systems*, San Jose, California, June, 1988, pp. 104-111.

[20] Zhou, S. "LSF: Load sharing in large-scale heterogeneous distributed systems", *Proceedings of Workshop on Cluster Computing*, 1992.

[21] Newman, H. and Mount, R. "Particle Physics Data Grid", *Proposal to DOE HENP*, 2000.