LSBATCH: A Distributed Load Sharing Batch System

Jingwen Wang, Songnian Zhou, Khalid Ahmed, and Weihong Long

Technical Report CSRI-286
April 1993

Computer Systems Research Institute
University of Toronto
6 King's College Road
Toronto, Ontario, Canada
M5S 1A1

The Computer Systems Research Institute (CSRI) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is an Institute within the Faculty of Applied Science and Engineering, and the Faculty of Arts and Science, at the University of Toronto, and is supported in part by the Natural Sciences and Engineering Research Council of Canada.

LSBATCH: A Distributed Load Sharing Batch System

Jingwen Wang, Songnian Zhou, Khalid Ahmed, and Weihong Long
Computer Systems Research Institute
University of Toronto

Keywords: load sharing, load balancing, batch processing, remote execution, distributed computer systems, parallel computing, heterogeneous computer systems.

Abstract

Batch processing, a primary mode of computing in mainframes and supercomputers, is becoming important for networked systems as the computing environments become more and more distributed. In this paper, we discuss the architectural and design considerations, and some important implementation issues of LSBATCH, a distributed batch system for large–scale, heterogeneous computer systems. LSBATCH supports batched submission and execution of parallel as well as sequential jobs in a system of up to several thousand hosts with possibly different architectures, Unix operating system varieties, and power. Implemented on top of the UTOPIA network operating system [5] as a distributed utility, LSBATCH takes advantage of the rich set of resource and load information services and efficient remote execution mechanisms of UTOPIA and provides scheduling algorithms to fully utilize the computing resources scattered around a distributed system. To the user, LSBATCH appears very much like a flexible single host batch system; the complexities of distributed and heterogeneous computers and system failures are hidden in the implementation. To satisfy real world constraints to resource sharing, a variety of configuration mechanisms are provided in LSBATCH to allow sophisticated site management policies to be established and enforced.

[1]

1 Introduction

With the dramatic improvement in microprocessor performance and networking technology, distributed computing is increasingly replacing mainframe- and supercomputer-based central-

[1]Authors' address: CSRI, 6 King's College Road, Toronto, Ontario, CANADA M5S 1A1, {jwang, zhou, ahmedk, long}@sys.toronto.edu

ized computing. Across various industries, the production workload running on mainframes and supercomputers is being migrated to distributed server machines and workstations. Batch processing, by which non-interactive jobs are submitted to the system to be processed at appropriate times, is an important mode of computing in a centralized environment. Although supporting only limited interactions with users (primarily through input and output files), batch processing has the advantages of allowing users to submit multiple jobs without having to wait for them to finish, and of giving the system the flexibility to schedule the jobs according to the jobs' priorities, and as computing resources become available. It is not surprising, therefore, that distributed batch processing support becomes important as distributed systems enter the main stream of computing.

The new issues in distributed batch processing center around the effective sharing of distributed and heterogeneous resources. Although the aggregate computing power of the servers and workstations may far exceed that of a supercomputer, the resources are scattered throughout the network, making effective use of them much more difficult than in a single host. Maintaining up-to-date information of loading conditions of the hosts for job scheduling is complicated. Moreover, the hosts are most likely heterogeneous, with different architectures, operating systems, or configurations (e.g., number and speed of their processors, memory size, and swap space), making the selection of the most suitable hosts for jobs hard. To complicate the matter further, the workstations are typically also used for interactive computing, and it may be unacceptable for batch jobs to impact interactive performance. In short, the tremendous opportunities for batch processing of compute intensive jobs in distributed systems are matched by the myriad of complexities and constraints involved.

In this paper, we present LSBATCH, a general purpose batch system for heterogeneous distributed systems. LSBATCH attempts to address the problems discussed above while providing a user interface very much the same as that of a traditional mainframe batch facility. LSBATCH is implemented on top of UTOPIA, a network operating system, which provides resource selection and transparent remote execution facilities for heterogeneous hosts [5].

In designing LSBATCH, we identified the following requirements of a general purpose distributed batch system.

- *single system image*: the distributed nature of batch job processing should be transparent to the users. Also, in a heterogeneous environment, although the resource requirements of the jobs should be specifiable by the users, selection of the resources best suited for the jobs should be transparent. Users care about getting their work done correctly and within their time constraints. Where to execute the jobs is the batch system's decision.

- *load sharing*: The job scheduling algorithm should take full advantage of the distributed resources in the system to optimize job throughput and turnaround time. Job placement should be responsive to resource load conditions and interactive activities on the hosts. Resource demands of batch jobs over a period may well exceed the total available resources. Therefore, scheduling of the batch jobs should accommodate conflicting user demands fairly.

- *sharing control*: Although load sharing is desirable, strong control on sharing, such as the kind "who can use which hosts during what time period under what load conditions for what kinds of jobs", should be configurable to allow local management and administration policies to be enforced and to ensure fair sharing. Without such control, the system would be unusable in most system environments as uncontrolled sharing leads to chaos or even wars.

- *heterogeneous operation*: a wide range of Unix platforms should be able to coexist in the same system, and share the batch job load as much as possible. In job placement, the varying degree of host power should be taken into consideration. Limited resource sharing in isolated parts of the system or multiple load sharing systems for individual Unix platforms, which we term *heterogeneous load sharing*, greatly diminishes the value of sharing.

- *very large scale operation*: for wide spread sharing of resources, the batch system should be able to support thousands of hosts, possibly geographically distributed. The system should scale up gracefully.

- *uniform support for all applications*: except for the restrictions on user interactions imposed by batch processing semantics, the system should make no assumption about the submitted jobs. Any non-interactive job that can be executed interactively should be executable as a batch job without change. Parallel jobs as well as sequential jobs should be supported, with no assumption about the programming paradigm used by the applications to achieve parallelism (e.g., message passing, distributed shared memory).

- *fault tolerance and availability*: in no case should a submitted (and accepted) batch job be lost. In the presence of a host crash, an error should be reported to the user who has a job running there, just as in the case of a job completion or abnormal termination. The system service should be available (i.e., responsive to user requests and perform the operations correctly) as long as one or more of the hosts are up.

- *security*: batch job execution should not be used as a means of getting around the system protection.

In the rest of this paper, we discuss the design and some important implementation techniques of LSBATCH to meet the above requirements. We will proceed top-down starting with a discussion of the system model and user interface in Section 2, followed by the system architecture in Section 3. The scheduling algorithms of LSBATCH is discussed in Section 4. Section 5 describes the support for parallel applications. We then briefly discuss the facilities in UTOPIA used by LSBATCH, and how batch processing interacts with the rest of load sharing system. Finally, a few concluding remarks are made in Section 7.

2 User's View

It is desirable that a simple model of batch job processing be presented to the users, with system implementation complexities completely hidden. The basic concepts of LSBATCH are *jobs*, *queues*, and *hosts*, with the hosts in the background. A user gets her work done by submitting batch jobs, either explicitly using a command provided by LSBATCH, or implicitly if the command is embedded in applications. Requirements for a job's processing may be specified at its submission time (see Section 2.3), including the queue to which the job is to join.

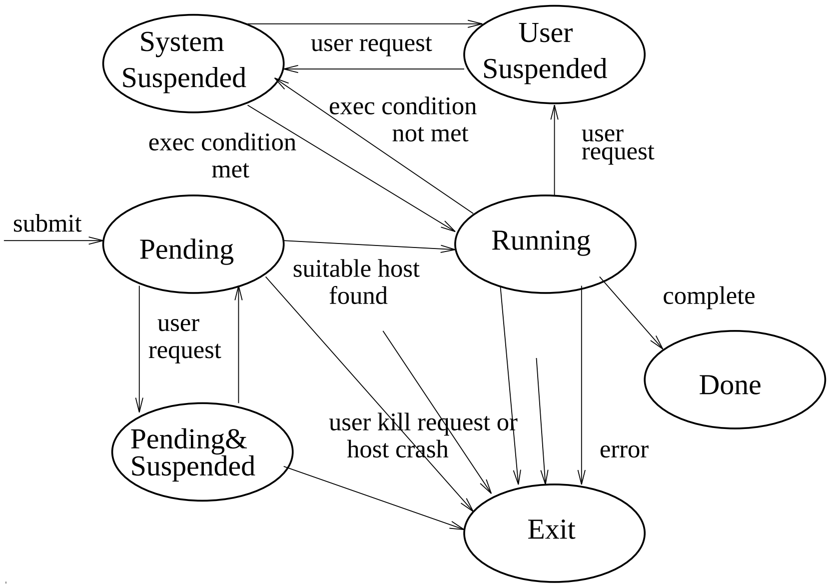

Figure 1: Batch job state diagram.

2.1 Batch Jobs

Figure 1 shows the state transitions that a batch job may experience during its life time, together with the conditions, or *job events*, that trigger the transitions. Upon being submitted to a queue, a job may stay in the *Pending* state for a while waiting for suitable host(s) to become available or for the time requirement for its execution (e.g., after 6 p.m.) to be satisfied. It then starts execution on the selected host(s) (*Running* state). While pending or running, a job may be suspended by its owner or the system manager (*Pending&Suspended* and *User Suspended* states), or by the batch system as the load situation on its execution host(s) changes (*System Suspended* state). A job suspended by the system may subsequently be suspended by its owner or the system manager, in which case it will have to be resumed by that person. Upon completion or error, a job goes to the *Done* or *Exit* states, respectively. We call a job residing in the batch system (somewhere along the path from *Pending* to *Done/Exit*) a *current job*. Any current job may be killed by its owner or the system manager. When a job terminates (entering *Done* or *Exit* state, i.e., becoming a *past job*) a mail message may be optionally sent to the user.

It is important to note that the flow of jobs through the various states is visible to the users upon inquiry, but is independent of the distributed nature of the system. In fact, we very much used a state diagram of a single-host batch system. This allows us to change the LSBATCH implementation for distributed system without changing the user's view of the batch system.

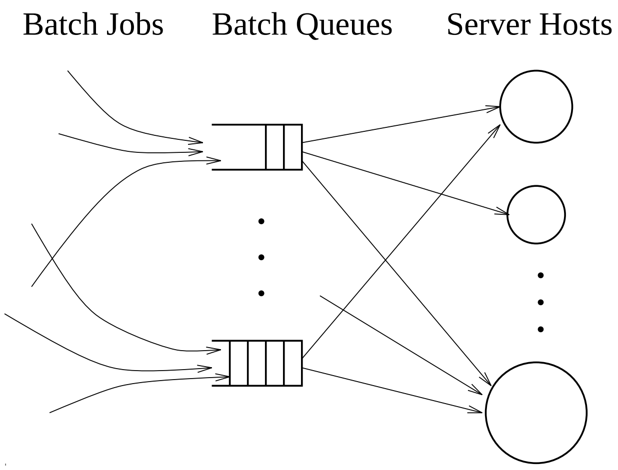

Figure 2: Job flow through LSBATCH system.

2.2 Job Queues

Unlike network batch systems such as NQS , DQS, or Condor [3], in which job queues are attached to individual or subsets of hosts, LSBATCH uses job queues purely as a job classification mechanism to provide different services to different kinds of jobs (see Figure 2). By default, jobs from a queue may be dispatched to any host for processing. This is consistent with the design philosophy of the UTOPIA system upon which LSBATCH is built: load sharing *by default*, with necessary restrictions.

Job queue is the main mechanism in LSBATCH to allow the system manager to configure the system to suit the processing needs of the workload with its various types of jobs. An arbitrary number of queues may be configured on site. Each batch queue may have a number of *queue attributes*, some of which are explained below [2].

priority: the priority of this job queue relative to others. At job dispatch time, the job queues are searched in the order of decreasing priorities such that jobs in a higher priority queue are considered earlier, and, if jobs in that queue are scheduled to run on some hosts,

[2] If a particular attribute is not specified, a default is taken.

1. *Priority* Queue:

```
priority=100; nice=10; currently enabled and inactive;
cpulimit=3600; exclusive jobs allowed;
run window:   5:18:00-1:7:30  19:00-7:30
users:     paul, john, bigshot
hosts:     all hosts
```

2. *Idle* Queue:

```
priority=10; nice=25; currently enabled and active;
cpulimit=userlimit; exclusive jobs not allowed;
run windows:    always open
users:     all users
hosts:     all hosts
```

Figure 3: Example batch job queues.

jobs in lower priority queues may have to continue pending. When it is necessary to suspend jobs, the system will also use queue priority such that low priority jobs running on hosts needed by high priority ones will be suspended.

nice: the reduction in UNIX process priority for jobs in this queue.

runWindow: time intervals during which jobs from this queue may be dispatched and run. Outside of the intervals, jobs will be suspended by default. The default is no restriction (i.e., 24 hours a day, seven days a week).

cpuLimit: the maximum amount of CPU time a job in this queue is allowed to consume, properly normalized to account for the different speeds of the processors. A job exceeding its cpuLimit will be killed by default. Similar resource limits may be specified for memory usage, and so on.

users: a list of user login names and user group names is specified so that only they can submit jobs to this queue. This queue attribute is useful for restricting access to the queue. Default is all users in the system.

hosts: a list of host names and host group names is specified so that jobs from this queue may only be dispatched to these hosts. Default is all hosts in the system.

exclusive: exclusive jobs (i.e., a job that should be executed by itself on a host) can be submitted to this queue.

In addition to the queue attributes, a job queue has two variables describing the current state of the queue:

- *enabled*: whether the queue is accepting jobs.
- *active*: whether the queue is processing jobs.

Figure 3 shows two queues, one high priority (*Priority*), one low priority (*Idle*). The high priority queue is accessible to only some of the users, and may be used only during weekday nights and weekends (thus, the queue is shown to be *inactive* at any time outside its run windows). The CPU time of each job is limited to one hour (normalized). It is possible for a *Priority* job to be dispatched to a host having an *Idle* job running, forcing it to be suspended. The queue priority decides the order of job suspensions.

2.3 User Interface

The distributed nature of LSBATCH is largely transparent to the users; therefore, the set of batch operations available to the users is essentially the same as that of a single–host batch system. The user interface should be powerful as well as easy to use. we achieve a desirable balance by using extensive options for flexibility and customization, and reasonable defaults to make it usually unnecessary to specify most of the options. Below are the names and semantics of some of the user commands.

bsub: submit a job (executable or shell script), sequential or parallel, with optional specification of the job queue, resource requirement, input and output files, time requirement, etc.

bjobs: obtain job information, such as job status, parameters, and reason for pending or suspension.

bpeek: inspect output of a job so far.

bhist: inspect the "history" of a job, e.g., when it is submitted, dispatched, suspended, resumed, completed, etc. Also available is summary information such as how long a job has stayed in a queue, has been executing, or has been suspended, and the amount of resources a completed job consumed.

btop, bbottom: move a job to the beginning or end of a queue, respectively. A user can only change the relative queueing positions of her own jobs, whereas the system manager can change any job's position.

bswitch: move a job from one queue to another.

bstop, bresume, bkill, bchpt, bmig: suspend, resume, terminate, checkpoint, or migrate the execution of one or a specified set of jobs, respectively. LSBATCH takes advantage of the kernel supported facilities for job checkpoint and migration available in some of the Unix systems, such as ConvexOS and UNICOS. Supporting job checkpoint or migration at the user level severely constrains the types of jobs that can benefit [3].

bsig: send a specified signal to one or more jobs. The job may be caused to checkpoint if application-specific checkpointing is provided.

bqueues: obtain job queue information.

bhosts: obtain host configuration and dynamic load information (such as the number of batch jobs dispatched to each host, and their current status).

busers: obtain information about users, such as the number of jobs they have outstanding.

bacct: generate accounting statistics by hosts, users, queues and/or time periods.

bqc: control the operation of a job queue — enable/disable and activate/deactivate a queue.

breconfig: reconfigure the LSBATCH system after some change to the LSBATCH configuration files.

3 System's View

As a distributed utility built on top of UTOPIA, LSBATCH follows the clustered architecture of UTOPIA [5]. Hosts in a large distributed system are organized into *load sharing clusters*, which usually follow the departments or work groups of the human organization being supported by the system. Figure 4 shows the clustered organization of an LSBATCH system, together with the detailed structure of a typical cluster.

3.1 Architecture within a cluster

Within each cluster, a centralized approach is used to process all the batch jobs. Batch jobs from all hosts in a cluster are submitted to a master batch server, MBD, which is responsible for scheduling the jobs to be executed on suitable hosts and keeping track of their progress. MBD is assisted by a slave batch server, SBD, on each server host (i.e., a host configured to process batch jobs), which provides the mechanisms for job execution and manipulation, and monitors the status of its local jobs on behalf of its master. MBD always runs on the same host as the master LIM of the cluster (see [5] for information on LIM structure). MBD obtains resource and current load information of all the hosts from the master LIM. When it identifies a host as being suitable for executing a sequential batch job, it dispatches the job to the SBD on that host, which starts the job immediately. For a parallel batch job, MBD would dispatch it only when enough number of suitable hosts can be found.

From our experiences with the UTOPIA system running at user sites with hundreds of hosts, the centralized management approach within the clusters coupled with the directed graph approach among the clusters maximizes opportunities for the applications and improves system scalability. Having MBD as the master (analogous to having the master LIM in the underlying UTOPIA system) ensures that all the system configuration and dynamic resource load information is available for making scheduling decisions. Similarly, all the current batch jobs are kept track of by MBD so that, whenever resources become available, jobs can be

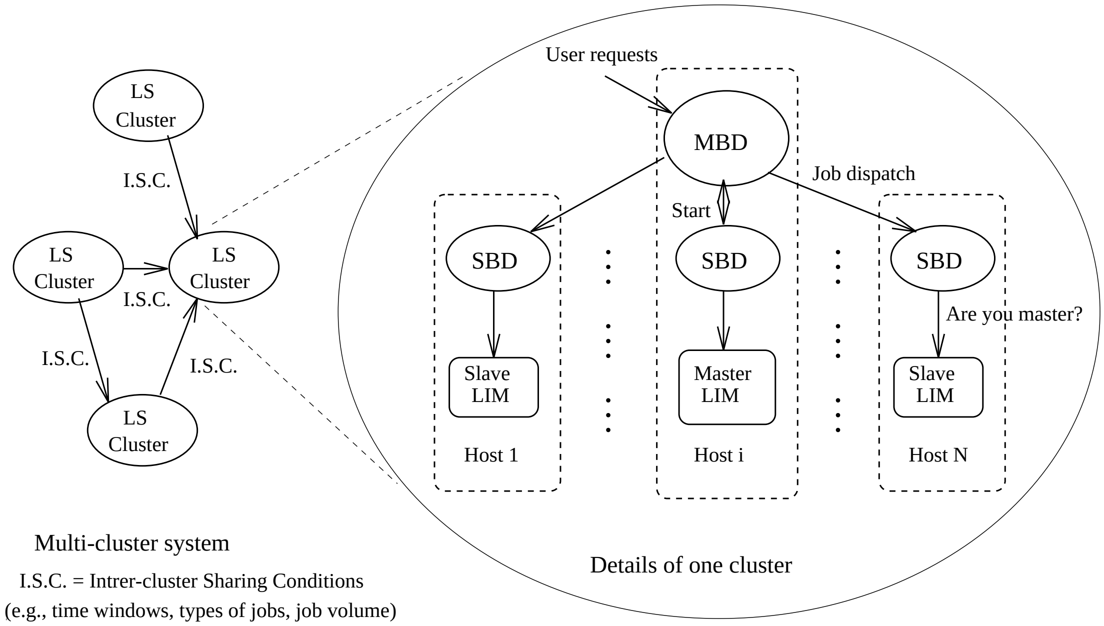

Figure 4: Architecture of the LSBATCH system.

dispatched. The MBD also keeps history information to ensure fairness in resource allocation and to minimize scheduling overhead. For example, it may be desirable to ensure that several user groups each get its fair share of computing resources from a set of server machines over a period of time (e.g., hours or days). The complete state information kept by MBD makes scheduling decisions optimal and responsive.

A different approach is taken in the Condor system [3], in which jobs are submitted to a server on the local host. A central *coordinator* server periodically wakes up and goes through two rounds among all the hosts participating in load sharing. In the first round, it decides whether each of the hosts is available for remote jobs (hence the load index is binary valued) and identifies all the pending jobs. It then enters into a second round to assign jobs to available hosts for execution. Similar to the Network File System (NFS) [4], no state information is kept by the coordinator. Acquiring such information on the fly requires that the coordinator communicates with all the potential server and client hosts, even if many of them do not have resources or jobs to provide. For a large system, the overhead may be substantial. Also, maintaining history information becomes difficult.

One consideration for maintaining no state information in the central server and queueing jobs at local servers is fault tolerance. Somehow, distributed algorithms are often considered more fault tolerant. Our experiences, however, show that this is not necessarily the case. A simple election algorithm is implemented for the LIMs in UTOPIA such that, if the current master goes down, a new master will come up quickly [5]. The configuration and load information is then quickly recovered. For LSBATCH, we piggyback MBD election on LIM election by having the SBD on the new master LIM host starting a new MBD (see Figure 4). All the state information about all current jobs is logged to files and is restored in-core at MBD boot time. Fault tolerance is in fact higher using such a stateful, centralized approach. In Condor, only local servers keep track of the locally submitted jobs, and, if the local hosts go down (say, workstations being turned off), these jobs will not be able to participate in job scheduling. In LSBATCH, jobs are submitted to the MBD, and may still be dispatched and processed while the submission hosts are down. Conceptually, jobs are submitted to *the batch system*, and should be processed as long as suitable execution hosts may be found; individual host failures should be masked as much as possible. Similarly, in Condor, if the coordinator's host goes down, service becomes unavailable, and pending jobs remain pending. An election algorithm similar to that in UTOPIA can be employed to solve this problem, but then a main reason for using a distributed algorithm no longer stands.

In terms of the merits of maintaining state information in the master server (MBD or coordinator), an analogy may be drawn between load sharing and distributed file systems. In the latter, stateful file servers provide more opportunities for performance optimizations and reduce network traffic. Similarly, for sophisticated batch job scheduling algorithms, keeping in-core state about jobs and their history seems to be crucial for long term scheduling; storing such information in files and reading it in on demand incurs high overhead.

3.2 Interactions among clusters

A further motivation for the centralized algorithm in LSBATCH is to facilitate the cooperation among the clusters – having a master, MBD, in each cluster makes intercluster load sharing effective because only the masters need to interact with each other to process all jobs. Such considerations go beyond computer systems to human organizations; in a large community, suborganizations with appropriate leaders or representatives who interact with each other on behalf of their constituents become necessary.

Batch job processing among the clusters is facilitated by the underlying UTOPIA system through a directed graph specifying the pattern and conditions of job flow between the clusters (see Figure 4). Distributed batch job processing among multiple clusters is motivated by the desire to share computing resources in a very large scale system. Conceptually, the scheduling of batch jobs should be performed in a centralized fashion to ensure fairness of resource allocation and various system– and user–specified rules for resource sharing, such as user resource quota. In reality, however, completely centralized operation would not scale up to very large systems, so partially distributed algorithms must be used that nonetheless are able to enforce rules based on dynamically changing system and user state information.

In Figure 4, the nodes are the load sharing clusters and the edges specify the desirable flow of jobs between them. The edges are configurable — they are present only if both clusters incident with it are configured to enable job flow. Furthermore, Inter-cluster Sharing Conditions (I.S.C.), such as the eligible types of jobs, the volume of job flow, and the time periods during which jobs can flow, can be configured. The multicluster structure is not only necessary for system scalability, but also desirable for sharing in a multi-departmental environments where sharing between departments is still desirable, but has to be more controlled, and relative independence and autonomy of the clusters are important. The continued operation of individual batch clusters is independent of each other for simplicity and isolation of failures.

3.3 Layered architecture

As a general purpose distributed batch system, LSBATCH is built in a layered and modular fashion, analogous to the underlying UTOPIA system. The structure is as shown in Figure 5. The base of the system consists of the MBDs and SBDs. Batch job service is accessible through LSBLIB, an Application Programming Interface (API). The set of batch commands described in Section 2.3 is built on LSBLIB. Alternative command interfaces, such as xlsbatch, a Graphic User Interface (GUI), and a set of NQS compatible commands are also built on LSBLIB. LSBLIB provides separation of batch user interfaces from the core of the system.

It is interesting to note the similar layered architectures of UTOPIA [5] and LSBATCH.

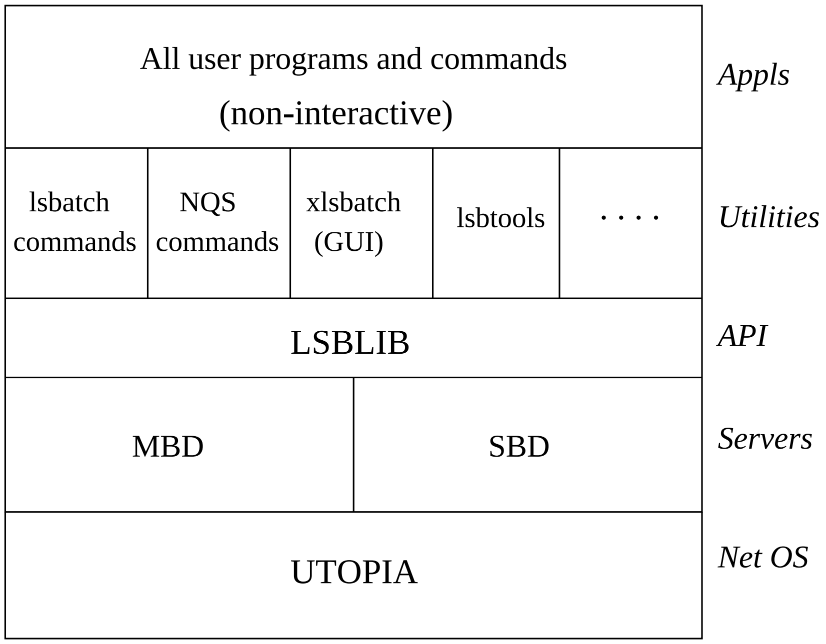

Figure 5: Layered software architecture of the LSBATCH system.

In many environments, a sequence of batch jobs are interrelated by data dependency, I/O requirements, or job sequencing. For example, some preprocessing of seismic data may be required before the main computation can start, and the results need to be output to a device such as a plotter in a subsequent step. The computation may itself consists of multiple components interrelated in a graph pattern. We call a set of interrelated batch jobs a *batch program*. LSBATCH supports the execution of batch programs by providing job status and sequencing in the servers, complimented by shell programming tools, lsbtools, analogous to the load sharing tool kit, lstools, in UTOPIA.

4 Job Scheduling

Scheduling in LSBATCH is implemented in MBD in the form of job dispatching, and in SBD in the form of selectively suspending and resuming jobs. Together, they attempt to maximize resource utilization throughout the system yet enforce the sharing policies established by system management.

4.1 Constraints to resource sharing

In an ideal world (like utopia), pure load sharing with no differentiation of users, jobs, and hosts would suffice. In real world distributed systems, however, a myriad of constraints in the forms of heterogeneous system base (including hardware speed and compatibility, and software availability), different user privileges, job characteristics and requirements, and fair arbitration among jobs in the case of resource contention have to be taken into consideration in scheduling batch jobs. This is not to mention the political and organizational considerations present in most resource sharing situations. LSBATCH, being a batch system for production use in diverse environments, strives to define a general framework for resource sharing that is simple to set up, administer, and use, yet powerful and flexible for customization. Before discussing the scheduling algorithms implemented in LSBATCH, let us first consider the various constraints to resource sharing built into the system:

- *time constraints*: certain jobs may only execute at certain times. This is implemented in three places. First, when a job is submitted, the start time and termination time may be specified. Secondly, each queue may have runWindows only during which jobs in the queue may be executed. Thus, these jobs are prevented from interfering with other jobs (batch or interactive) outside of the runWindows. Similarly, runWindows may be configured for individual hosts. Lastly, LSBATCH manager may control a job queue by allowing/disallowing the acceptance and processing of jobs of the queue.

- *user access constraints*: certain job queues may be accessible to only some of the users. Such queues may be configured with high priority, or users in one department may have preferential access to hosts in their own department (when used in combination with host access constraints below).

- *host access constraints*: certain job queues can only use a subset of the available hosts. This may be desirable if certain applications are only licensed to run on those hosts, or if jobs in the queues should not be allowed to run on arbitrary hosts, disrupting other activities. In addition, the number of jobs concurrently residing in a host is limited by default or by the configurable values. User may explicitly require that a job be executed on one or more specific hosts or on hosts selected from a candidate set. The host loading conditions under which jobs can be dispatched and run, or should be suspended (e.g., due to interactive activities) are specifiable for individual hosts (see Section 4.3 for more information). This is useful, for example, to prevent batch jobs from affecting interactive performance. Finally, some queues allow jobs to be executed exclusively on their hosts. No other job, batch or interactive, will be routed to the hosts by UTOPIA or LSBATCH during the exclusive jobs' execution.

- *resource consumption constraints*: the amount of any type of resource a job may be allowed to consume (such as CPU time, memory space, /tmp storage, swap space, disk I/O) can be specified and enforced.

In addition, since LSBATCH is implemented on top of UTOPIA, all the constraints to load sharing in effect in LIM are automatically enforced. For instance, a host may be locked by LSBATCH manager, its owner(s), or by certain applications that should be executed there exclusively. A host may be configured with special resources to limit its accessibility to applications.

4.2 Scheduling in MBD

The main functions of MBD are 1) to process user requests, 2) to keep track of all the current jobs, 3) to dispatch jobs for execution when appropriate, and 4) to coordinate with the local SBDs and the remote MBDs. MBD ensures that all the scheduling constraints listed in the previous section are satisfied when jobs are dispatched.

The scheduling algorithms of LSBATCH attempts to send jobs to hosts that can execute them correctly and rapidly, given the architectural, operating system, and other resource requirements supplied by the batch job submission, while maximizing the system throughput, and satisfying the time constraints for job completion. The underlying UTOPIA provides a rich collection of load indices for the resources of each host, such as CPU queue lengths averaged over various periods of time, CPU utilization, the amount of available user memory, paging rate, disk I/O rate, the number of users logged in, the amount of time the host has not been used interactively, and the amount of swap and /tmp disk space. MBD considers all such information together with the resource requirements of the jobs in making dispatching decisions. Most of the job scheduling intelligence of LSBATCH is located in MBD, which acts as a "traffic cop" to direct the flow of batch jobs. Upon receiving a job, the MBD holds it in its queue until the job's time requirement (e.g., after 6 p.m.) is met and a host that meets all the job's resource requirements and has low enough load becomes available [3]. This policy avoids the undesirable situation of an SBD receiving a job from MBD only to decide that it is not ready to execute the job (e.g., current load is too high), and consequently puts the job in its local queue. While the job is stuck in this queue, another host may become available. The centralized and delayed scheduling policy used in LSBATCH proves to be desirable.

Job placement criteria for batch jobs are typically different from those for interactive jobs. The former are more concerned with longer term load, as batch jobs are expected to run for a while. MBD obtains load information from LIM and selects suitable hosts itself, rather than using the task placement service of LIM, which is intended for interactive jobs.

An important requirement of some production environments is to share a set of server hosts among several groups of users such that, over a period of time (e.g., an hour, a day, a

[3] The thresholds for the load indices on each host for accepting batch jobs may be individually configured.

week), resources are allocated to jobs belonging to each user group according to configurable shares (e.g, 2:3:2 among three groups). This is a generalized, macroscopic version of the fair sharing scheduling implemented in supercomputer operating systems such as UNICOS and ConvexOS. We call such a set of shared hosts a *host partition*, which may be a whole load sharing cluster, or a part of it. LSBATCH supports configurable host partitions. When resources are not contended for, the shares are not enforced (i.e., everybody to his/her needs). When jobs start to queue up, however, the shares are approximately adhered to in MBD's job dispatching policy using recent history of resource consumption of the user groups.

4.3 Scheduling in SBD

When a job is dispatched to a host, it starts immediately as MBD has already ascertained that the host is ready. The SBD on the host receives from MBD, along with the job description, directions for managing the job, such as 1) the priority of the queue to which the job belongs, 2) the resource consumption limits provided by the user or the queue, and 3) the queue's runWindows and the start time and termination time of the job optionally provided by the user. SBD uses such information together with its *scheduling vector* to determine whether jobs should be stopped, resumed, or terminated. High and low "watermarks" for each of the load indices provided by LIM (see Section 4.2) may be optionally set in the scheduling vector such that, if the value of a load index goes over its low watermark, no job will be dispatched to run there and no already suspended job will be resumed until the value falls below the low watermark. Similarly, a job running on this host will be suspended if the value of any of the load indices goes beyond its corresponding high watermark. It is required that the difference between the high and low watermarks be large enough so that, as host load fluctuates, frequent job suspensions and resumptions are avoided. The watermark values of CPU load indices are *unnormalized* to reflect the number of jobs running.

For exclusive jobs, SBD would request the local LIM to lock the host to prevent any job, batch or interactive, from being directed there. When the job is over, the host is unlocked.

It may be desirable to suspend a low priority job to allow a newly arrived high priority job to run first. For example, running a few short jobs in the middle of one long job execution provides quick turn around time for the short jobs without hurting the long job much. This effect can be achieved *automatically* by configuring the high priority queue to have preemption privilege over lower priority queues.

5 Parallel Application Support

Parallel processing is an important means of achieving high performance for compute intensive applications. The large number of hosts in a distributed system makes this possibility attractive. Unfortunately, little software support is currently available for parallel applications running in workstation clusters.

Two types of support software are needed. The first type provides mechanisms for communication and synchronization among the tasks of a single parallel application. We call such support *intra-application support.* The computing paradigm used may be message passing, shared memory, or derivatives of them. PVM [2], Network Linda [1], and Express are some examples of software for intra-application support. The second type of software provides support for allocating processors to parallel applications, starting the tasks of applications, and controlling the applications (e.g., suspension, resumption, and termination). We call such support *inter-application support*, because it relates to the interactions among parallel applications and with the environment. Inter-application support should be provided by a network operating system such as UTOPIA. Thus, in this paper, we are primarily concerned with the inter-application support of parallel applications.

Currently, parallel application packages use rather simplistic methods for selecting processors to run individual applications. For example, PVM uses a user-managed *host file* to indicate the list of hosts to run an application on; tasks are initiated on the hosts in a round-robin fashion. While such a scheme may be adequate for a dedicated environment, most real world systems are multi-user multiprogrammed. If two applications of different users use overlapping sets of hosts, contention for resources (e.g., CPU, or even worse, memory) may result, leading to much reduced performance for both applications (and probably two unhappy users). This partially defeats the purpose of parallel computing. Worse yet, such compute intensive applications may be started on workstations supporting interactive users, leading to unacceptable interference. Clearly, for parallel computing on shared workstations to be generally effective, the Parallel Virtual Machine has to be mapped to the physical machines based on resource configuration and current load conditions. In other wards, such packages should be made load sharing-aware. For starting remote tasks of an application, mechanisms such as Unix *rsh* are typically used. The delay in starting a number of tasks one by one can be tens of seconds or minutes. For relatively fine grained applications, such overhead severely limits the speedups of the applications.

Load sharing software such as UTOPIA addresses the inter-application support aspects identified above.

5.1 Interface and mechanisms

In designing parallel application support in UTOPIA and LSBATCH, we decided that the support must be independent of the particular parallel applications and packages, yet be able to support a wide variety of them with minimal efforts from them. This is consistent with the design goals of UTOPIA as being a general purpose network operating system, and of LSBATCH as being a general purpose distributed batch system. No assumption about the applications should be made. Furthermore, as with sequential applications, we require that the same parallel applications be able to execute either interactively or in batch mode without change to their binaries. Our approach may be contrasted to that taken in DQS, a distributed queueing package in which an option "–pvm" for the batch job submission command provides a tight marriage between DQS and PVM.

Figure 6 shows the architecture of parallel application support in LSBATCH. A parallel batch job is submitted in the same way as a sequential job, except that the initial number of processors required may be specified. The MBD would queue up the job the same way as for

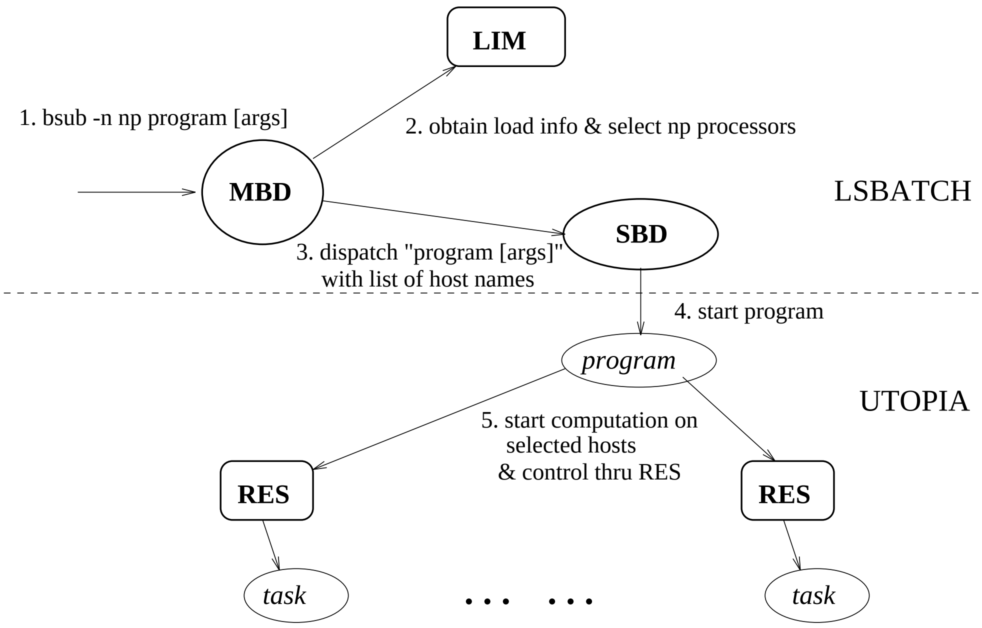

Figure 6: Support of parallel applications in LSBATCH.

a sequential job. At scheduling time, if enough number of processors (some of which may be located on the same multiprocessor hosts) are available, the job would be dispatched to the first host selected, together with the list of selected host names.

The SBD on the first host would start the application (or, to be precise, the *root task* of the application) in the same way as for a sequential application. However, the SBD will be aware of the list of hosts being used by the application, and, if any of the hosts' load exceeds its configured scheduling vector values, it will, by default, suspend the whole application by sending an appropriate signal to the application.

From the point of batch job startup, that is, once a parallel job crosses the dotted line in Figure 6, the parallel application support in the underlying UTOPIA system takes over from LSBATCH. Job control operations, such as suspension, resumption, and termination, will be performed by LSBATCH on parallel jobs the same way as for sequential jobs. The root task of the application uses a generic interface to decide whether it is started interactively or by LSBATCH by checking for an Unix environment variable, LSB_PARAHOSTS, which the SBD sets with the list of selected host names. For interactive execution, the application will have to contact LIM by itself to obtain execution host names.

A parallel application may select among several levels of support in UTOPIA. At Level 1, only the list of host names, either obtained through LSBATCH or directly from LIM, are used by the application; remote tasks are started on those hosts by the application using mechanisms outside of UTOPIA (e.g., *rsh*). At Level 2, UTOPIA's Remote Execution Servers (RES) on the remote hosts are used to start the initial remote tasks there to take advantage of the low overhead of RES. The remote tasks then disassociate themselves from UTOPIA.

For job control, signals would have to be propagated to the remote tasks by the application itself. Level 3 is the same as Level 2 except that the distributed job control mechanisms and remote execution transparency in UTOPIA are taken advantage of by the application. Job suspension, resumption, termination, and checkpoint operations, as well as all Unix signals, are propagated throughout all tasks of the application by UTOPIA without the application being aware of. Finally, in Level 4, all remote tasks, not just the initial ones on the remote hosts, are started by RES. This is the full parallel application support by UTOPIA. The level of support suitable for each application or package depends upon its needs and nature. In the next section, we examine a few examples.

5.2 Example parallel applications and packages

To assess the generality of LSBATCH's support for parallel applications, let us examine how it can be used in the three cases below.

5.2.1 Lsmake

This is a load sharing version of GNU make that runs on multiple hosts to speed up the make process [5].

It is submitted to LSBATCH in the following form:

```
bsub [-n numproc] [-R resReq] [other bsub options]
                        lsmake [make options and targets]
```

Lsmake is started by the SBD on the first selected host. Lsmake interprets the makefile, and uses RESes to start all its remote tasks; it uses Level 4 support.

This example is applicable in general to other master-slave type of applications, with the lsmake program being an example of a master, and whatever tasks it creates being slaves. Slaves interact with the master through signals and tty input and output, but not among themselves.

5.2.2 lsPVM

Each host participating in a PVM application has to have a PVM daemon, pvmd, for the user running the application. We modified the code of pvmd and PVM library to enable load sharing in both interactive and batch modes using the same binaries [4]. A load sharing PVM (lsPVM) application is submitted to LSBATCH in the following form:

```
bsub [-n numproc] [-R resReq] [other bsub options] lspvmd appl [args]
```

The master PVM daemon, lspvmd, will be started by the SBD on the first selected host. The master in turn starts slave lspvmds on the other selected hosts through RESes. Since

[4] There are substantial differences in Version 2.4 and 3.1 of PVM; our discussion here applies to the 3.1 version.

job control is already provided in PVM, the slave lspvmds disassociate themselves from the RESes upon startup (that is, Level 2 service is used).

The original PVM requires that the daemons be started beforehand, and *rsh* is used. With load sharing, by which the execution hosts are selected dynamically rather than based on a static list of host names in a file, it becomes generally unacceptable to start the daemons on all potential hosts. Moreover, multiple lsPVM applications by the same user are allowed to coexist on the same host (possibly a multiprocessor). Dynamic startup of the daemons becomes necessary. The low overhead of remote parallel execution using RES makes dynamic startup of the daemons usually not a performance concern. For example, starting daemons on 25 hosts takes about 4 seconds [5]. Once the slave daemons are started, the master daemon would then start "`appl [args]`" locally to kick off the application execution.

The support of Linda [1] is quite similar to PVM; Linda also uses a daemon process on each of the participating hosts, except that the daemons are started at the beginning of application execution. Thus, to make Linda applications load shared, less changes to the package compared to PVM are needed.

5.2.3 TCGMSG

Like PVM, TCGMSG is a communication package for parallel applications, but it does not use user daemons. In a modified version of TCGMSG supporting load sharing, a command, *lspload*, is used to start up an application. An unmodified binary TCGMSG application, `appl`, is submitted to LSBATCH in the following form:

```
bsub [-n numproc] [-R resReq] [other bsub options] lspload appl [args]
```

Upon starting on the first selected host, lspload starts a communication server process, gserv, and gets back the port number upon which *gserv* listens for connection requests. *lspload* then starts "`appl [args]`" on all selected hosts using RES and passes gserv's port number (and host name) to them. The instances of `appl` then call back to *gserv* to establish contacts.

For both TCGMSG and PVM, no change to the application is necessary for batched execution, although the message packages need to be modified or extended for load sharing (in both interactive and batch modes).

5.3 Compatible interface with interactive use

All the parallel applications and packages may be used both interactively and through LSBATCH. The options, however, are slightly different.

As an example, an interactive invocation of lsmake looks like:

```
lsmake [-n numproc] [-R resreq] [make options and targets]
```

The `-n` and `-R` options are used by lsmake in the interactive mode, but not in the batch mode. Instead, they are supplied to the **bsub** command.

Similarly, for PVM and TCGMSG, an interactive invocation is also slightly different from batch mode:

```
lspvmd  [-n numproc] [-R resreq] appl [appl arguments]

lspload [-n numproc] [-R resreq] appl [appl arguments]
```

Such minor differences seem to be unavoidable and tolerable.

The above structure for supporting parallel applications also works for a few other packages that we and others have tried, including proprietary packages whose design and implementation we know little about. Our experiences to date suggest that the parallel application support in LSBATCH and UTOPIA are indeed general purpose.

5.4 More dynamic applications support

In the above discussion, the set of hosts used by a parallel application is selected by the MBD *before* the application starts. The application simply goes ahead and uses the hosts in the way it wants. Some of the hosts may become overloaded during the job's execution, but it is undesirable to suspend the whole job. For an application that dynamically creates tasks, the set of hosts it uses should desirably change dynamically to both exploit the available resources and to avoid interfering with other applications.

UTOPIA supports dynamic allocation/deallocation of processors using a generic interface which is an extension of LSLIB, UTOPIA's API. In this interface, a pool of processors currently allocated to the application is maintained. The interface allows the application to explicitly change the size and host composition of the pool at specific points during the computation. Alternatively, the pool size and host composition may be updated periodically in a manner transparent to the application. This is useful for applications for which program correctness is independent of the degree of parallelism (e.g., lsmake). Placement decisions for remote tasks only consider the current processor pool. In addition, the decision of whether the application is executing interactively or in batch is made internally, so that the LIM or MBD, respectively, may be contacted to acquire/release processors.

For parallel applications requiring resources in multiple clusters, the management of the processor pool may involve interacting with multiple LIMs or MBDs. While the concept of cluster is beneficial from the viewpoint of scalability and administration, it is desirable to present the application programmer with a uniform view of the entire system.

6 Relationship between LSBATCH and UTOPIA

For relevant information about UTOPIA discussed below, please refer to [5].

LSBATCH makes direct use of two components of the underlying UTOPIA system: LIM and LSLIB library. LIM is used for locating MBD, for deciding on which host to start MBD, and for getting resource load information. All programs built on LSBLIB, such as lsbatch commands and xlsbatch, use LSLIB to communicate with the local LIM in order to locate MBD and to obtain the resource requirement information stored in LSLIB's task database. MBD uses LSLIB to communicate with LIM.

Note that the remote execution facility in UTOPIA, implemented in LSLIB and RES, is not used in LSBATCH. RES supports highly transparent remote execution, but does not interfere with the execution of current tasks. In contrast, in batch system, no interaction between batch job and the user who submitted it occurs, so the high degree of remote execution transparency is unnecessary. On the other hand, extra control and monitoring of the jobs are needed, which should be implemented as a higher level facility, such as SBD, rather than being lumped into the general purpose remote execution mechanism RES. Moreover, an agent on each host is necessary to ensure the availability of the MBD. It is possible to use LIM or RES to start MBD, but this violates the principle that the common load sharing system base, UTOPIA, be unaware of applications and utilities running on top of it. Independent of LSBATCH, parallel applications may use RES for remote task execution.

Sharing of heterogeneous hosts for batch processing comes *for free* due to the underlying UTOPIA system's support. At job submission time, the job's resource requirement is either specified by the user using command line argument or looked up in the task database managed by LSLIB. Among these requirements may be information about the host hardware and software requirement information. Without inspection and interpretation, MBD forwards such resource requirements of a job to the master LIM, which selects eligible hosts accordingly and only returns load information of qualifying hosts for MBD to make the last stage selection.

There are a number of network batch queueing packages, such as NQS/Exec, DQS, and Condor [3]. Some of the services provided by LSBATCH are similar to those in such packages. However, the approach and architecture of LSBATCH is unique. Instead of being a standalone package, LSBATCH is built on top of a network operating system, UTOPIA, and shares a rich collection of services as discussed above. This makes LSBATCH more sophisticated in job placement and more fault tolerant. Most of the other queueing packages have limited view of the system environment; the information available to the scheduler may include host availability, the number of batch jobs on each host, Unix load average, and the presence of interactive activities. Fault tolerance appears to be absent in these packages. For example, the host of the central server is usually selected manually, and, if it goes down, services become unavailable. A more important implication of the integrated approach to load sharing in UTOPIA as compared to the standalone packages is that all processing activities, interactive or batch, initiated by any program (such as lsmake, LSBATCH, or lstcsh, a load sharing Unix command shell), are coordinated through LIM such that they do not make conflicting job placement decisions. For example, if a batch queueing package and a network make program operate independently, they may send jobs to the same hosts, resulting in conflicts and poor performance.

7 Concluding Remarks

In this paper, we discussed the design and implementation issues of LSBATCH, a load sharing batch system for large, heterogeneous distributed systems. We adopted to a simple, modular design for LSBATCH, following the design approach of the underlying UTOPIA network operating system.

LSBATCH and UTOPIA currently run on a wide variety of Unix platforms, such as SunOS, HP-UX, IRIX, ConvexOS, AIX, Solaris, Ultrix, and DEC OSF/1. With a highly modular architecture and well-defined kernel interface, porting the systems to other Unix platforms is relatively easy. Porting to some of the more recent, non-Unix operating systems is expected to be quite feasible. LSBATCH has been in production use at a number of sites supporting dozens to hundreds of hosts each. Our experience indicates that the system requirements we identified are well met, and that the system is robust and stable.

We are currently working on enhancing LSBATCH's support for the parallel applications and batch programs, and conducting research into scheduling algorithms for complex workload running on the target types of systems for LSBATCH — very large scale and heterogeneous distributed systems. Such algorithms are made complicated not only by the heterogeneous and complex workloads and system environment, but also by the myriad of constraints on resource sharing. With LSBATCH as a testbed, and with our accessibility to real world computing systems, we are able study many of the difficult problems that are impossible to dream up in a pure academic environment.

Acknowledgments

This work has been partially funded by Bell Northern Research Inc., by the Natural Science and Engineering Research Council of Canada, and by a University Research Incentive Fund from the Government of Ontario. We are grateful to Dave Wright, Craig Fischberg, Greg Gaus, and Albert Miele, Jr. of Pratt & Whitney Co. for their valuable input to the design of LSBATCH and feedback on the experience in using UTOPIA and LSBATCH in their systems.

References

[1] C-linda user's guide and reference manual. Technical report, Scientific Computing Associates, Inc., 1993.

[2] A. Geist et al. Pvm 3 user's guide and reference manual. Technical report, ORNL/TM-12187, Oak Ridge National Laboratory, May 1993.

[3] M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. In *Proceedings of the* 8^{th}*International Conference on Distributed Computing Systems*, pages 104–111, San Jose, California, Jun 1988.

[4] R. Sandberg and et al. Design and implementation of the sun network file system. In *Proceedings of USENIX Conference*, May 1985.

[5] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software: Practice and Experience (to appear)*, also as Technical report, CSRI No. 257, University of Toronto, April 1992.