System Management for IRIX 6.5

MPI, Array Services, Miser, NQE, Checkpoint/Restart

NQE Development Team
Strategic Software Organization
Silicon Graphics, Inc.

ABSTRACT: *MPI, Array Services, Miser, NQE, and Checkpoint/Restart are key components for IRIX system management. This paper will describe these components and the interactions among them.*

Overview

The key to system management is schedule, execute, and control jobs so that they run in an efficient manner. This is provided in IRIX 6.5 by the following components:

MPI	execute multi-process and multi-node jobs.
Array Services	manage multi-process and multi-node jobs. Array Services is an important component in running and managing MPI jobs.
Miser	schedule jobs in a manner that will deliver fast response time for interactive jobs and fast turn-around time for batch jobs. Miser partitions memory and CPU according to administrative groups of users, manages the active workload, and minimizes cache hits and swapping.
NQE	schedule, route, monitor and control the job backlog on a single node or a cluster of nodes.
Checkpoint/Restart	checkpoint and recover jobs during system shutdown or failure. The jobs may be running on a single machine or on a cluster of machines managed by Array Services.

This paper will describe these components and the interactions among them.

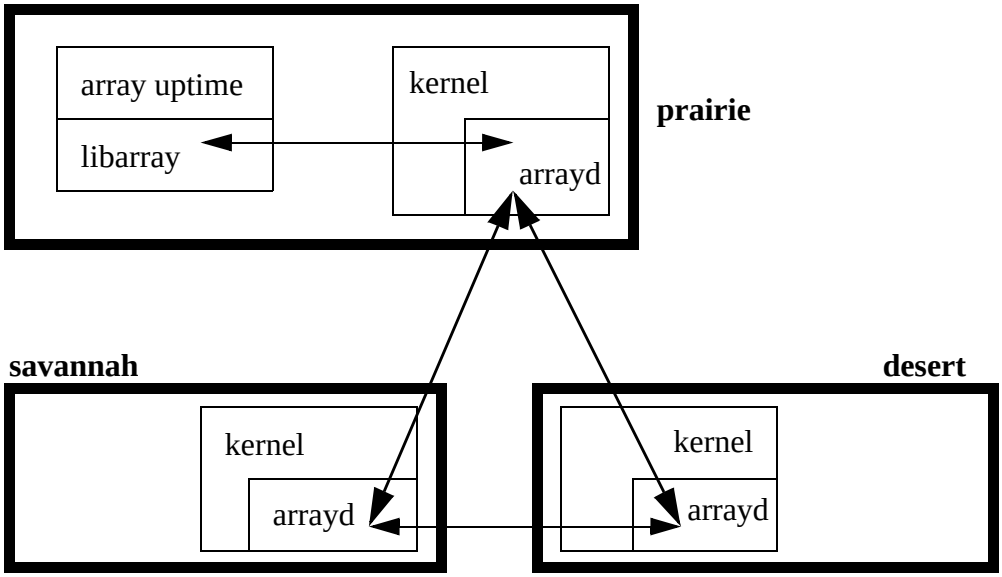
Array Services

Array Services provides management of multi-process and multi-node jobs, as well as execution of commands across a cluster. Each job is given a global Array Services Handle (ASH) to treat processes as a single entity across the cluster. Accounting data can summarize the resource usage of an entire login session or batch job, significantly cutting down the amount of disk space required to store the accounting data.

```
prairie% array uptime
  desert:     up 3 days, 6:43, 73 users, load average: 18:21, 16.57, 16.43
  savanna:    up 10 days, 2:03, 27 users, load average: 4.99, 4.03, 3.72
```

```
  prairie:      up 1 day, 15:41, 18 users, load average: 5.20, 5.94, 7.76
  forest:       COMMAND FAILED
                array services not available
prairie%
```

FIGURE 1. Array Services



Array Services is made up of four major components:

1. A number of client commands such as the array command illustrated above.
2. A user-state server, called arrayd, running as root on each node in the cluster.
3. Support in the IRIX kernel, in the form of array sessions.
4. A library, called libarray, which encapsulates the client/server and user/kernel interfaces.

All communication across the array occurs in the arrayd daemons. Most of the client commands are simply wrappers for libarray: they parse the command line options, bundle them together into a request, invoke a function from libarray, and then print the results. In a few cases the libarray function will simply invoke a kernel function. More commonly, however, libarray will open a connection to an arrayd server and send the request to it for processing. The server may either process it by itself or forward the request to one or more servers in the cluster for processing. Once the request has been handled, the results are sent back to the client process, where libarray transforms them into a format suitable for the client. In addition, users are free to write client programs of their own using libarray.

The client commands are:

array	Runs a specified “array command” on each of the nodes in the cluster. The array commands are configured by the system administrator.
ainfo	Obtains configuration and status information about clusters, machines, servers, and array sessions.

akill	Sends a signal to all of the processes in an array session, or to a process on a remote machine.
arshell	Executes the specified command on the specified host in a manner similar to the rsh command, propagating array session information and resource limits.
aview	Uses a Motif-based GUI to show the cluster and the jobs running in it.
ascheck	Verifies that all of the nodes in a cluster have compatible Array Services configurations.

MPI (and Array Services)

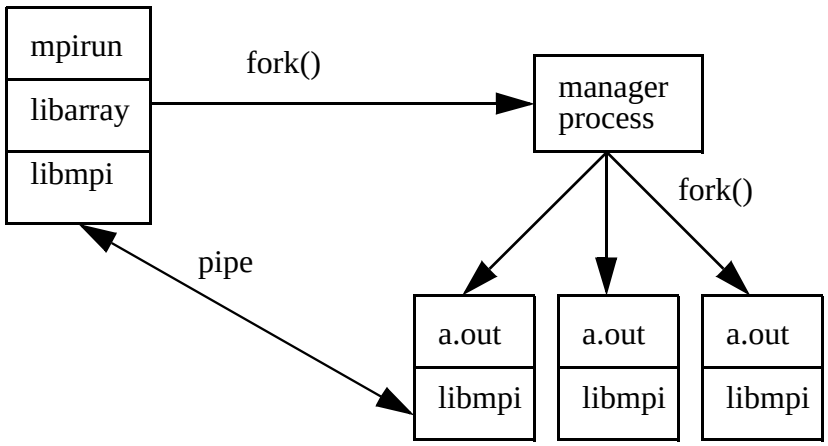
The Message Passing Interface (MPI) is used for high performance on massively parallel machines and on workstation clusters. The mpirun command specifies the number of processes to run and the hosts to run them on.

MPI 3.0 uses Array Services to launch all MPI applications on all hosts, including the local host. All processes in an MPI job share a global ASH and are controlled through Array Services. All the processes of an MPI job may be signalled or killed, or suspended and resumed, as one.

The following command specifies that three processes of a.out run on the local host.

```
myhost% mpirun -np 3a.out
```

FIGURE 2. MPI processes on the local host



See Figure , “MPI and Array Services,” on page 6 for information on MPI with remote processes.

Miser

Miser is an application to schedule and reserve CPU and memory resources of a machine, balancing the needs of interactive users and batch jobs. Different queues can be configured to partition the system among administrative groups.

Users specify a job's resource requirements and Miser determines when the job can run or if it can run in the requested timeframe. The following command requests that Miser schedule 64 cpus, 1 gigabyte of memory, and 12 hour of cpu time for the job a.out.

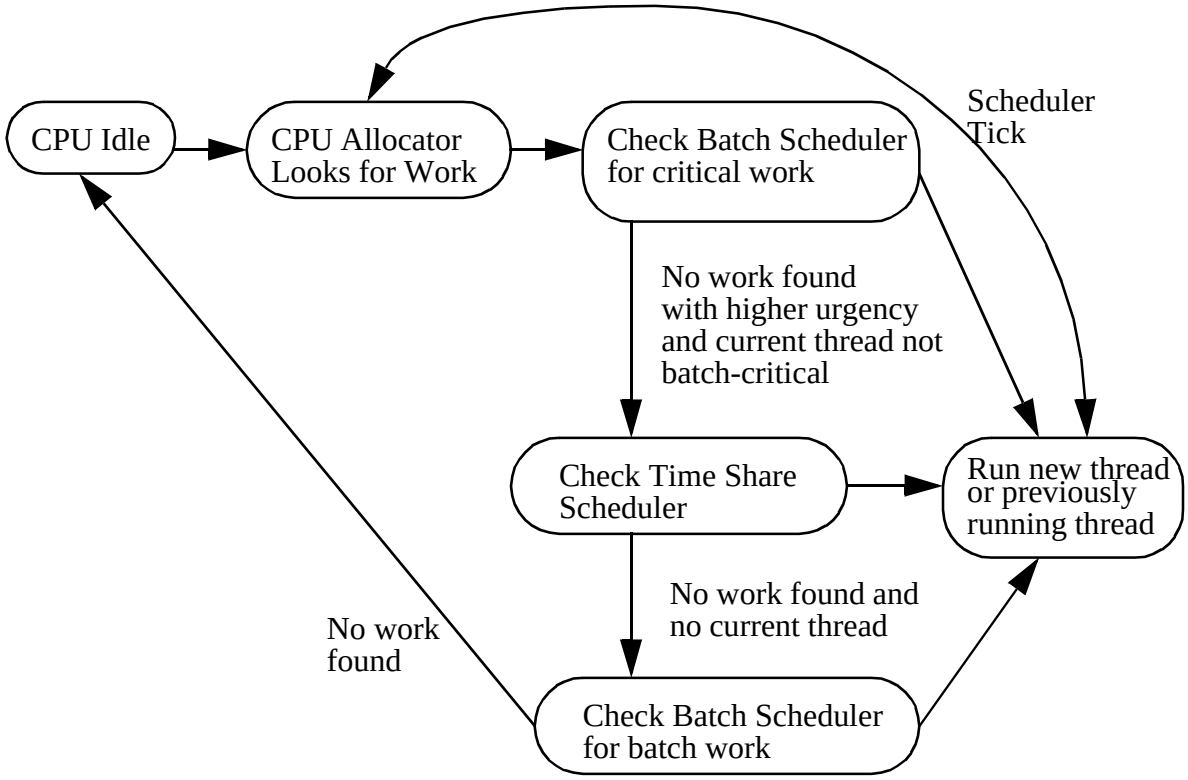
```
myhost% miser_submit -q marketing -o c=64,m=1g,t=12h a.out
```

Once Miser accepts the job, the job is guaranteed to have the requested resources at the scheduled time. As a result, a job does not have to compete for resources and should complete faster with more stable run-times. However, a Miser job may have to wait for its scheduled reservation period.

Miser assigns the job a start time and places it on the batch queue. At the time of a job's start time, the job is considered *batch-critical* and has the highest non-realtime priority. However, if a CPU becomes available before the start time, the job can run opportunistically.

Batch jobs not submitted through Miser and interactive jobs also run opportunistically, utilizing CPUs that have not been reserved. See Figure 3, "Scheduler State Diagram," on page 5. Memory, however, is partitioned and only jobs submitted through Miser will be allowed to use the memory resources configured for Miser.

FIGURE 3. Scheduler State Diagram



NQE

The Network Queueing Environment (NQE) is a workload management product that schedules, routes, monitors and controls the execution of jobs on a single host or a cluster of hosts. Users can submit their jobs to the NQE cluster, specify the resources needed for the job, and not worry about where the jobs will run. Based on the resources needed and the resources available, NQE will select an appropriate server, route the job request to that server, schedule and initiate the job request, and return STDOUT and STDERR files to the user.

Users may easily view system load, as well as submit, status, signal, and delete jobs using the nqe GUI.

System administrators may checkpoint and restart batch jobs through NQE on IRIX 6.5.

Users specify the needed resources indirectly by submitting to a particular queue or directly with options on the qsub or cqsub command. NQE will use the resource information to manage jobs by workload class. If a job exceeds its resource allocation, it is killed.

The following process limits are enforced by the IRIX kernel.

- per process core file size
- per process data segment size

- per process permanent file size
- per process stack segment size
- per process CPU time
- per process memory size
- per process working set size

The following job limits are enforced by NQE for IRIX. If a job exceeds these limits, there will be an entry in job log when the job is killed.

- per request CPU time
- per request memory size
- Number of CPUs

Note that the limit on number of CPUs includes background processes running in parallel as well as MPI jobs with parallel processing.

Checkpoint/Restart

IRIX Checkpoint and Restart (CPR) is a tool to capture an “image” or statefile of a running job and to restart it from that point at a later time. The default behavior is to kill the job when checkpointing it, but the job may also be allowed to continue running. A job may be running on the local host or on another host managed by Array Services. CPR may be used to enhance system availability, provide load and resource control or balancing, or to facilitate simulation or modelling.

The cview command provides an X Windows interface to CPR.

Checkpoint/Restart and MPI

CPR does not support checkpoint and restart of MPI jobs. The ability to checkpoint and restart single-system MPI jobs is planned for the second quarter of calendar year 1998. The ability to checkpoint and restart multi-system MPI jobs is planned for the third quarter of the calendar year 1998.

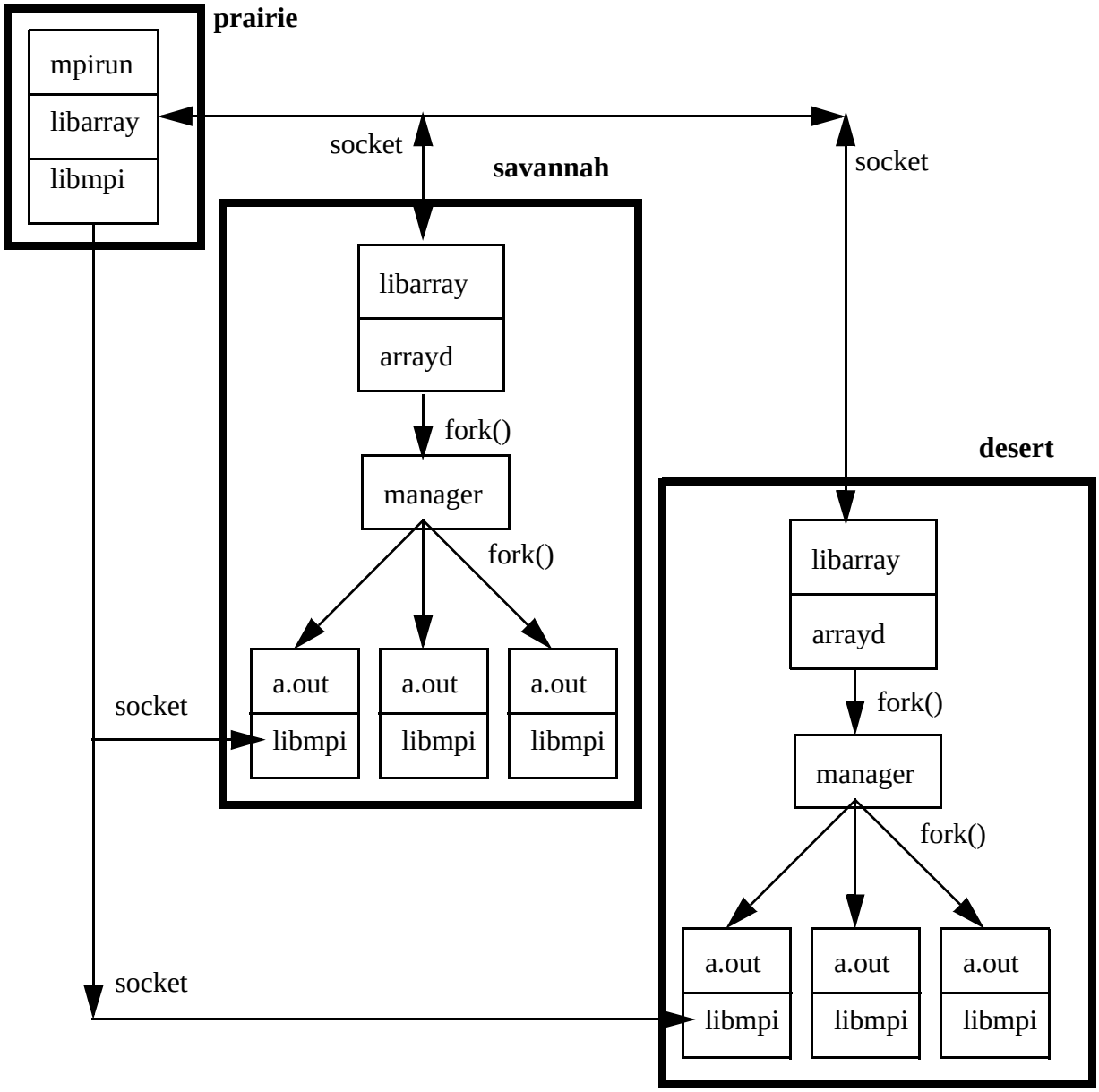
MPI and Array Services

MPI 3.0 uses Array Services to launch all MPI applications. MPI jobs with remote processes may be suspended and then resumed (but not checkpointed and restarted) using the global ASH from Array Services.

Communications for remote processes is accomplished with sockets. The following command will initiate mpirun on prairie with the MPI child processes on savannah and desert.

prairie% mpirun savannah,desert 3 a.out

FIGURE 4. MPI processes on remote hosts



NQE and Array Services

NQE assigns an ASH to each job. NQE batch jobs are visible in Array Services just as interactive jobs are. Array Services commands may be used in NQE batch jobs in the same manner as they are used interactively.

NQE and MPI

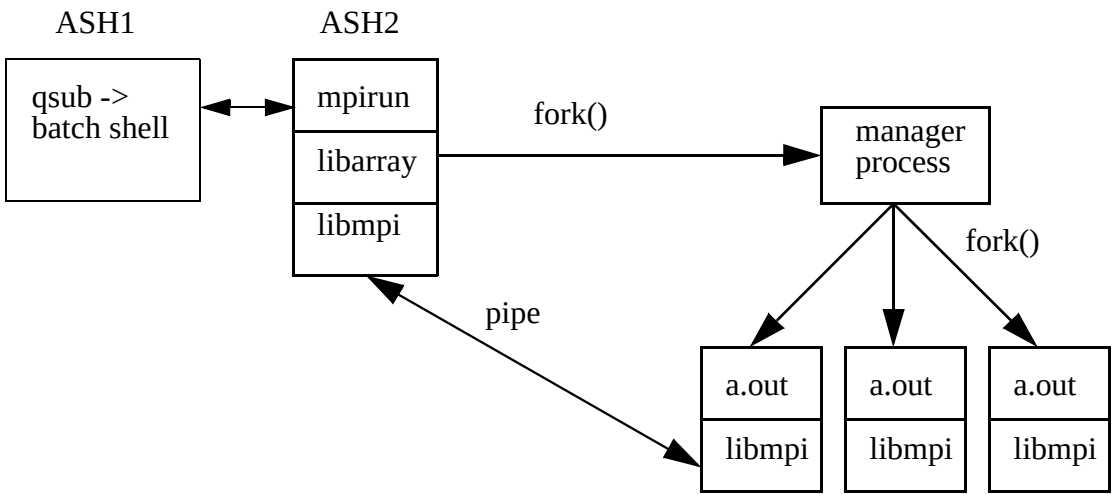
NQE only supports MPI jobs running on the local host. When NQE routes a job to a host, it expects that that is where the job will run.

If an MPI job with remote processes is initiated under NQE, job limits will only apply to local processes. However, deleting or signalling the job will delete or signal the remote processes. NQE supports IRIX checkpoint/restart, and will use CPR support of MPI jobs as those features become available.

To run an MPI job on the local host, use the -np option on the mpirun command:

```
mpirun -np 3 a.out
```

FIGURE 5. NQE and MPI on the local host



Note that NQE and MPI both create an Array Services ASH. The login shell for the batch job will have a different ASH than the MPI job itself.

NQE and Miser

NQE manages workload by ordering jobs based on their resource requirements. Jobs are queued until the resource usage of the jobs in a queue drops below a specified limit. For example, a site might have three queues, one for large CPU jobs, one for medium-sized CPU jobs, and one for small CPU jobs. Each queue has a limit of the number of jobs that may run simultaneously, allowing the site to create a job mix of small, medium, and large jobs according to its requirements.

In contrast, Miser defines administrative partitions and reserves resources for the group of users of a partition, designated by a Miser queue. Workload management is accomplished by reserving the CPU and memory resources so that they are never oversubscribed. A job submitted to Miser will receive a reservation of resources but may not actively run until sometime in the future.

NQE 3.3 has a new job scheduling type called Miser Normal that will submit jobs to a Miser queue. If Miser can't guarantee that the job will start in a configurable timeframe, NQE will keep the job in queued state and attempt to submit the job later. While the job is in queued state it may be moved to a different queue.

Once Miser accepts the job, NQE shows the job in run state. Miser may not start the job immediately but, once accepted, the job is guaranteed to start within the NQE configured timeframe.

An NQE queue may only submit to one Miser queue, but any number of NQE queues may submit to a single Miser queue. Thus, a set of NQE queues can be used to define different workload classes for a Miser queue.

Miser and Array Services

Miser only supports jobs running on the local host. There is no interaction between Miser and Array Services.

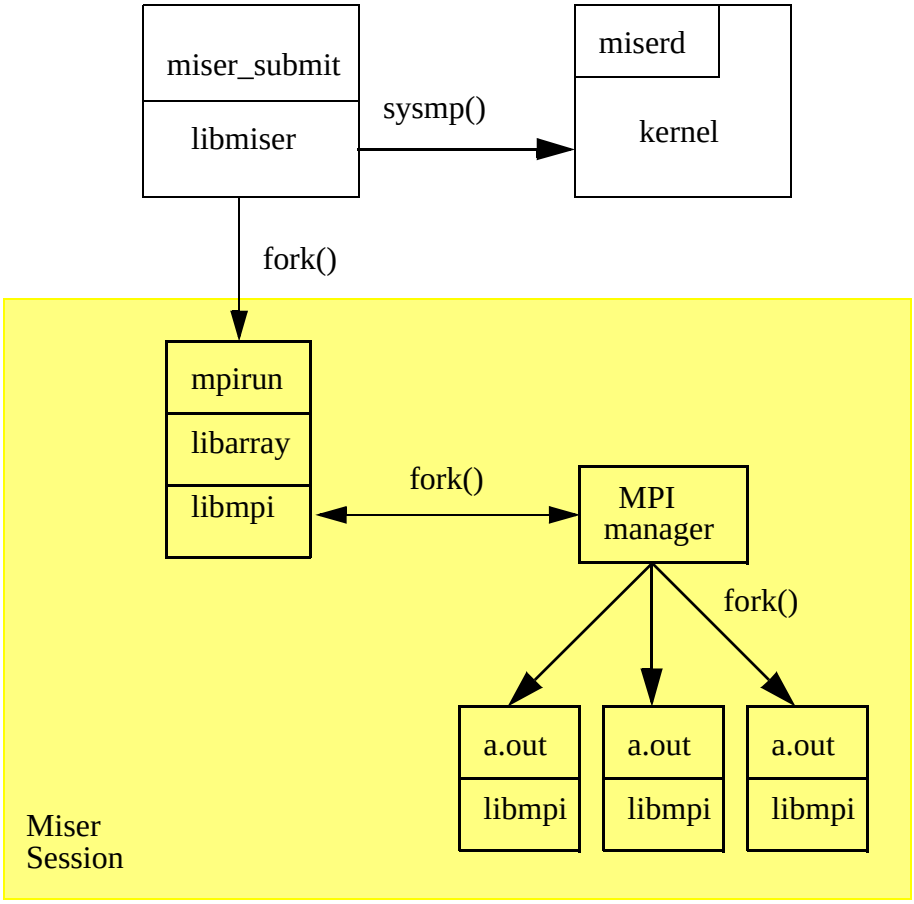
Miser and MPI (and Array Services)

Miser only supports single node MPI jobs. To run an MPI job on the local host under Miser, use the command:

```
% miser_submit -q engr -o c=8,m=1g,t=12h mpirun -np 8 a.out
```

Miser places the forked mpirun process in a batch run-queue and will move it to batch-critical at the scheduled time. When the process is scheduled, the process and all its children inherit the Miser resources, contained in a resource set. Miser requires a parent-child relationship for processes to share a resource reservation. See Figure 6, "Miser and MPI on a single node," on page 10.

FIGURE 6. Miser and MPI on a single node



NQE, Miser, MPI and Array Services on a single host

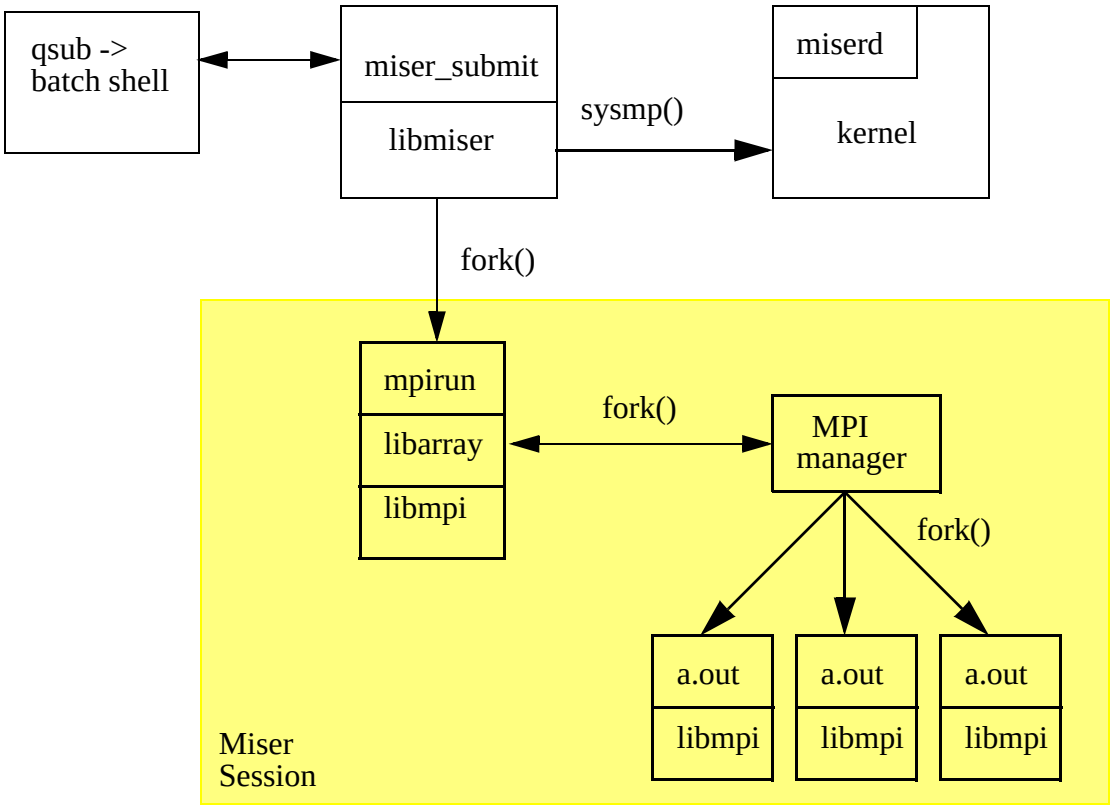
MPI jobs may be submitted to NQE to use Miser scheduling, as long as all the MPI processes run on the local host. All the processes of an MPI job may be signalled or killed, or suspended and resumed, with a single NQE command for each action. See Figure 7, “NQE, Miser and MPI on a single node,” on page 11.

NQE submits to a Miser queue and controls the job.

```
% cat mpijob
#!/bin/csh
miser_submit -q q1 -o c=5,m=1m,t=1h mpirun -np 3 a.out

% qsub -q miser_q1 mpijob
```

FIGURE 7. NQE, Miser and MPI on a single node



NQE, Miser, MPI and Array Services on a cluster of hosts

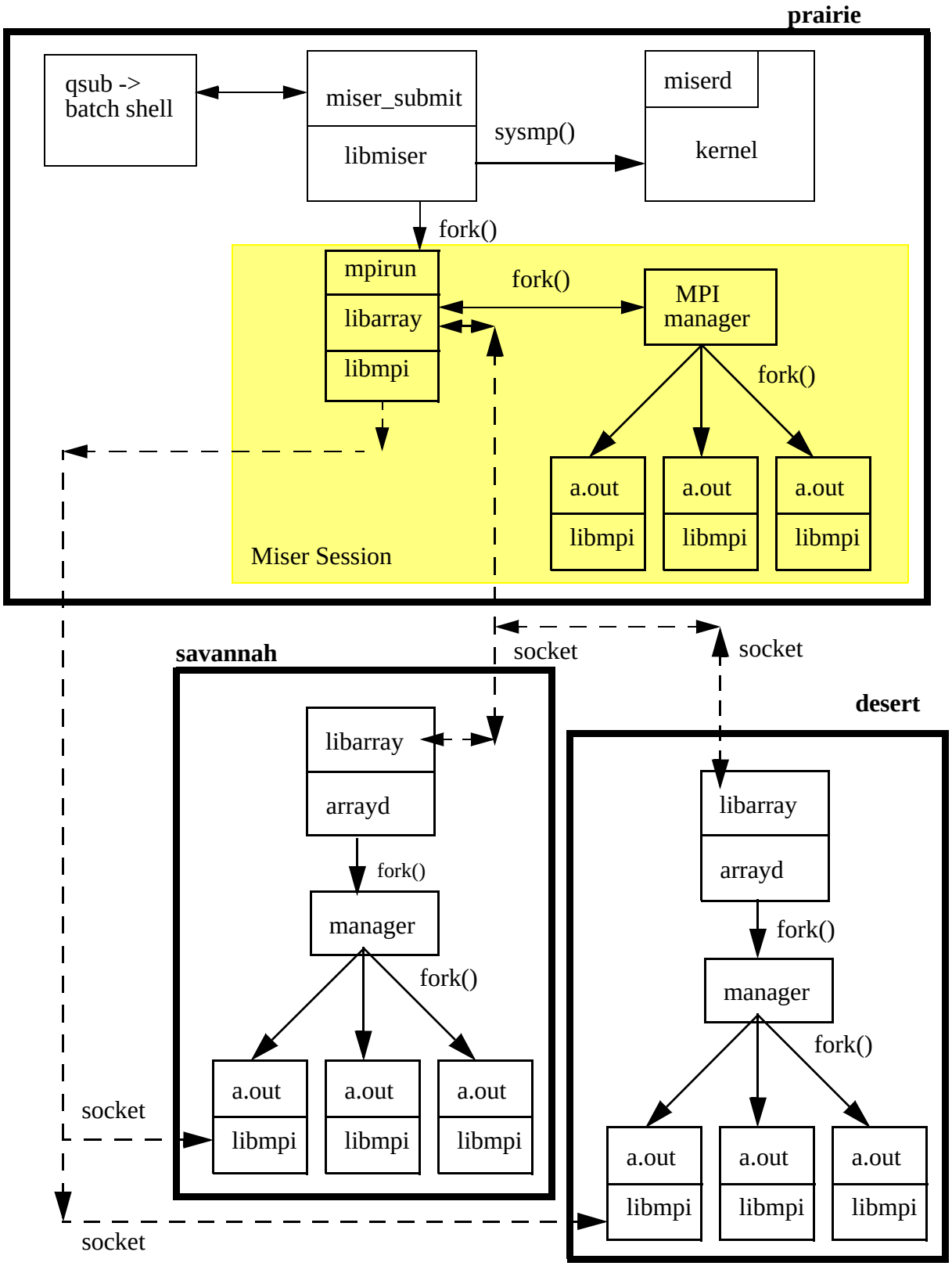
MPI jobs using Miser under NQE will have the same problem as MPI jobs using Miser started from an interactive process: there is no way to coordinate the resource reservation among hosts. The parent-child relationship is lost between the local and the remote processes. Similarly, any NQE limits checks will affect only processes on the local host. In addition, the batch job could be using an NQE run slot on the local host while waiting for processes to initiate on the remote hosts. See Figure 8, “NQE, Miser and MPI on a cluster of hosts,” on page 12.

Notice that in the following example, the Miser Session is only on the local host.

```
% cat mpijob
#!/bin/csh
miser_submit -q q1 -o c=5,m=1m,t=1h mpirun prairie,savannah,desert 3 a.out

% qsub -q miser_q1 mpijob
```

FIGURE 8. NQE, Miser and MPI on a cluster of hosts



Road Map

Silicon Graphics is committed to solving the problems of high performance and high availability for distributed computing. The following time line shows the progression of completed and planned features.

Released Features

4Q1996	Irix 6.4: 32PE support NQE 3.1: initial Origin support CPR 1.0: initial cpr release array 3.0: 8x32 node
1Q1997	NQE 3.2: 64 bit limits, cpr support
2Q1997	
3Q1997	Irix 6.4 update: 64PE support NQE 3.2.2: softjob limits cpr 1.1: pthreads
4Q1997	NQE 3.2.2: softjob limits
1Q1998	MPI 1.2: 64 PE support, LSF support IAUD: interactive limits
2Q1998	NQE 3.3: miser integration MPI 1.2.1: Miser, single system checkpoint Array 3.1: single machine fork/exec, single machine miser
3Q1998	Irix 6.5: 128 PE support FairShare II Miser CPR 1.2: fetchop, shmem Array 3.2: source merge, UNICOS cmds
4Q1998	MPI 1.3: 48x128 support, cluster cpr phase 1 Array 3.3: 48
1Q1999	UDB/Kernel limits Cray style accounting