UTOPIA: Load Sharing in Large-Scale Heterogeneous Distributed Systems

Songnian Zhou

Computer Systems Research Institute
University of Toronto
and
Platform Computing Corporation

zhou@sys.toronto.edu

Workshop on Cluster Computing
Florida State University
December 3, 1992

What's a Distributed System?

Load Sharing

Redistributing system workload among hosts to improve
. performance and resource accessibility.

Motivations:

- balancing the loads on hosts
- exploiting more powerful hosts
- allowing access to non-local resources
- supporting parallel & distributed applications
- one-system view — virtual multicomputer

Heterogeneity:

- Configuration — e.g., CPU speed, memory size, disk space
- Operating system — e.g., AIX, OS/2, HP/UX
- Architecture — e.g., MIPS, SPARC

Large-scale systems: 1000s of hosts

Why?

- Powerful hosts scattered around
- Large, hungry parallel applications

UTOPIA: Load Sharing Facility

A load sharing system for large, heterogeneous
. distributed systems

Requirements:

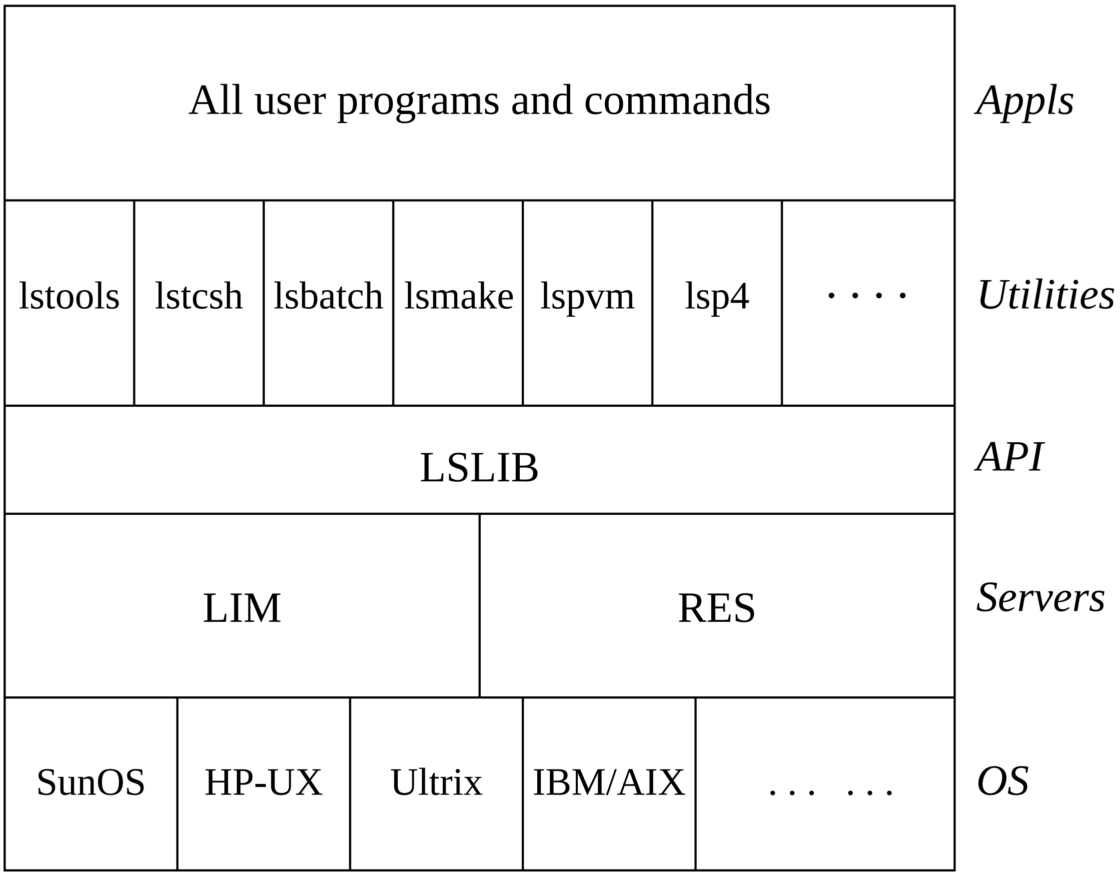
- transparent remote execution for *all* applications
- no need to change applications
- exploiting heterogeneity — architectural, OS & configurational
- responsive to tasks' resource needs
- scalable to 1000s of hosts
- supporting autonomy, protection, and configurability
- supporting parallel & distributed applications

UTOPIA Design

Assumptions:

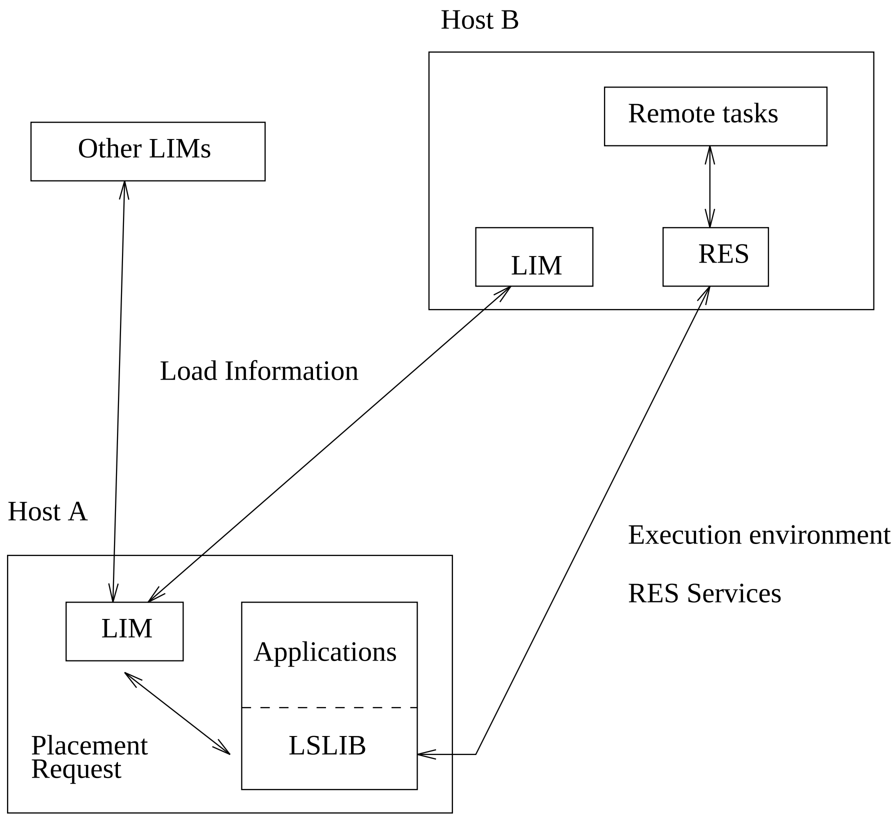
1. remote execution at task initiation time
2. distributed shared file space, with relaxations (e.g., NFS automount)

System architecture:



UTOPIA Design

System structure:



Load Information

- Average (scaled) CPU queue length & utilization
- Amount of user memory
- Average paging & disk block transfer rate
- Number of login sessions
- Host idle time
- Available swap space
- Available /tmp space
- Host status: `ok, busy, unavail, locked,`
  . `send/recv only, private, ...`
- Network load (as seen by the host)

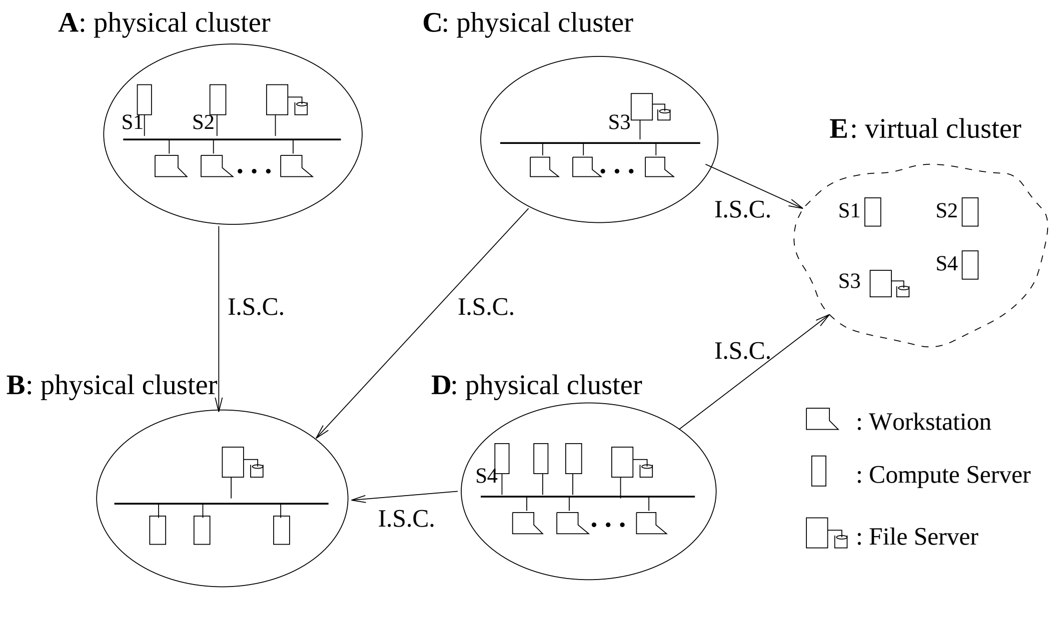
Used by LIM together with system configuration information

Load Information Distribution: Clustering

Within clusters: Centralized for efficiency — *CENTRAL*

Among clusters: Distributed for scalability

- *2-level hierarchy* can cover 1000s of hosts
- *Directed Graph* for selective inter-cluster sharing



I.S.C. = Inter-cluster Sharing Conditions (e.g., time windows, types of applications, job volume)

Task Resource Requirements

- *General Resources*: cpu, mem, disk I/O, swap, /tmp, login, ...
- *Special Resources*: whatever manager cares to associate with subsets of hosts

Examples:

`-R cpu:sparc:hspice`

— Lightly loaded/fast CPU of SPARC arch with Hspice licence

`-R idle=10:mem=12:swap=25:-ultrix`

— Non-Ultrix host with at least 12 MB mem, 25 MB swap, and been idle for 10 min or more

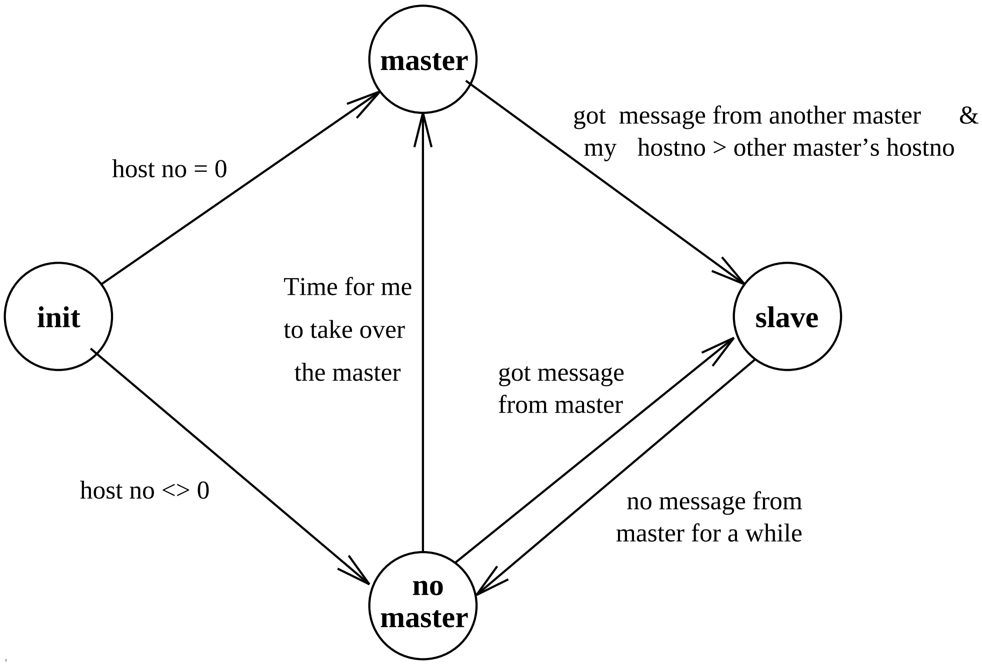
`-R mem:login`

— Machine with lots of memory and fewest users (for lslogin)

Tasks' resource requirements recorded in system & user *task eligibility lists* consulted automatically by LSLIB at task submission time

Master LIM Election & Locating

Simple election algorithm:



Master LIM locating:

1. try default master first, if no response
2. probe other hosts in turn

LIM Performance

1. Task Placement Delay:

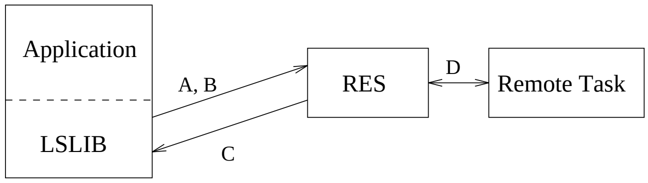
Number of candidate hosts	Number of hosts requested		
	1 host	5 hosts	10 hosts
15	3.8 ms	4.6 ms	5.0 ms
33	4.5 ms	7.7 ms	8.4 ms
60	5.8 ms	14.2 ms	18.5 ms

2. Master LIM CPU & Net Overhead:

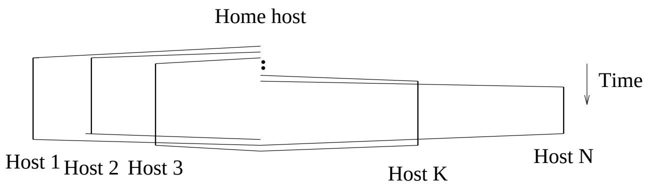
Load sharing cluster configuration	Master LIM CPU (% time)	Network traffic (KB/second)
15-host cluster	0.08%	0.04
33-host cluster	0.15%	0.09
60-host cluster	0.29%	0.17
Two (33, 27) clusters sharing each other	0.22%	0.37
20 50-host clusters 5-way sharing (emulation)	1.1%	1.2

(Measured on 60-SUN/IPC cluster during busy production periods.)

Transparent Remote Execution



- Simple, light-weight
- Multiple concurrent remote tasks easily supported
- Application-RES connection costs once
- Parallel & asynchronous operation to reduce delay
- Supports tty, signals, and user authentication
- Inter-operation among heterogeneous hosts



Delay (second):

Host pair	1-way cold	1-way warm	5-way cold	15-way cold	25-way cold
Sun IPC→Sun IPC	0.79	**0.10**	1.7	3.2	4.2
DEC→DEC	0.87	0.11	1.8	3.7	–
DEC→Sun IPC	0.92	0.15	–	–	–
Sun IPC→DEC	0.69	0.16	–	–	–

Transparent Remote Execution

Transparency: neither local parent nor remote child knows ...

- Maintain identical Unix execution environment
  - user & group IDs
  - default file creation mask
  - interval timers
  - signal masks
  - resource usage limits & stats
  - environment variables
  - current working directory
- Support terminal I/O using pty between RES & remote task
- Support Unix signals
- Same level of transparency achieved between heterogeneous hosts

Transparent Remote Execution

Authentication: Two solutions:

1. Privileged-ports: same as in *rlogin*, *rsh*

   - Applications using LSLIB must be *setuid*
   - In *ls_init*(), a number of privileged sockets created
   - Application immediately restores user's ID
   - Number of remote connections limited — one used, one less left

2. Authentication server: RFC931 Protocol, Kerboros

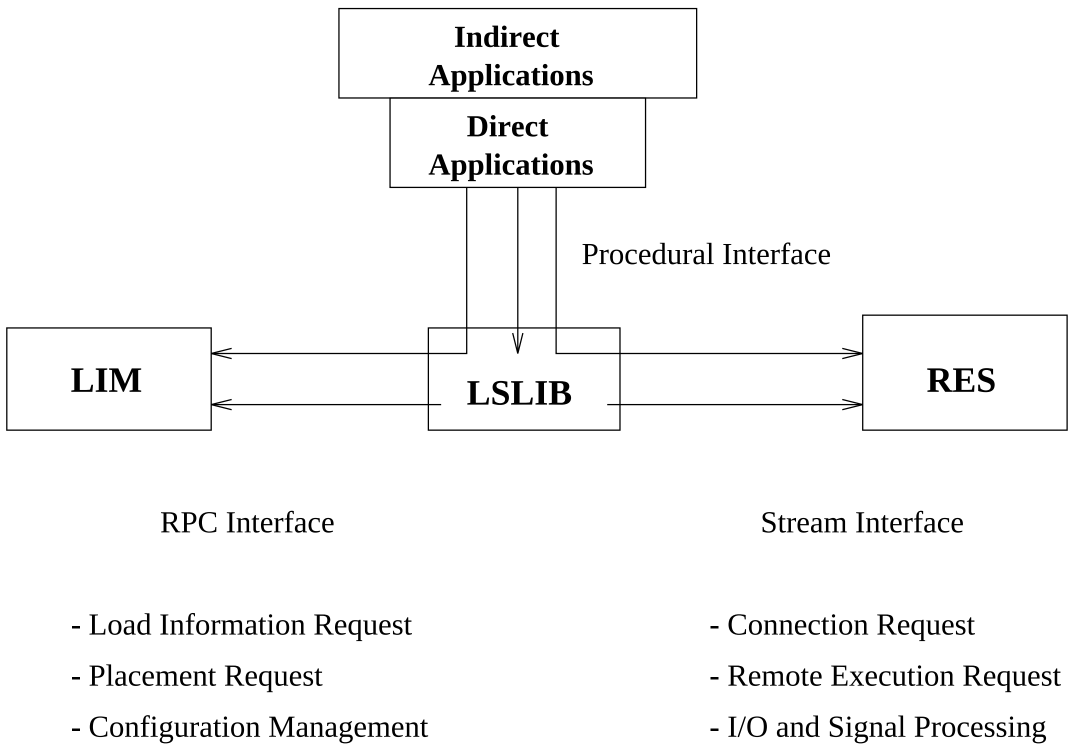
Multilingual applications: dynamic protocol selection

Applications Support

Two classes of applications

- *direct applications*: trucks and trains
- *indirect applications*: "free-loaders"

Load Sharing LIBrary (LSLIB): focal point of UTOPIA



Applications & Their Support

A Toy Application: a simple command interpreter

```
#include <lsf/lsf.h>
   ... ...
   ls_init();                        /* load sharing initializations */
   gethostname(myhostname, MAXHOSTNAMELEN);
   desthost = malloc(MAXHOSTNAMELEN);  /* space for best execution host name*/
   for (;;) {                        /* repeatedly read and execute commands */
      printf("Command>");            /* print prompt */
      read_command(commandName, arguments);          /*Read in a command line*/
      if (!ls_eligible(commandName, resourceReq)) {   /*trivial command?*/
         local = TRUE;                                /*local execution */
      } else {                       /*non-trivial command, ask LIM for best host*/
         ls_placereq(resourceReq, 1, &desthost, 0, 0); /*res. req. of the cmmd used*/
         if (strcmp(desthost, myhostname) == 0) {  /*local is best? */
             local = TRUE;
         } else {                    /*need to execute remotely on desthost*/
             local = FALSE;
             if (!ls_isconnected(desthost, socks)) /*connection not existent?*/
                 sock[0] = ls_connect(desthost);   /*set up initial connection*/
         }
      }
      pid = fork();                  /* create a child to execute the command*/
      if(pid == 0)  {                /* I am the child process */
         if (local == TRUE)          /* local execution ?*/
            execvp(commandName, arguments);
         else                        /*send command to desthost to execute */
            ls_rexec(socks[0], commandName, arguments, 1);
         printf("Execution failed\n"); /*child process should not return */
         exit(-1);                   /*execution failed; child process exits */
      } else {                       /* I am the parent process */
         wait(&status);              /* wait for child to finish */
      }
  }
```

Applications & Their Support

Types of Applications (and *examples*):

1. Interactive jobs: *Lstcsh* (Load sharing *tcsh*)
   - Full functionalities of *tsch*
   - Transparent remote execution of all Unix commands & user jobs
   - Distributed execution of jobs in shell scripts

2. Batch jobs: *Lsbatch* (Load sharing *batch*)

3. Parallel jobs

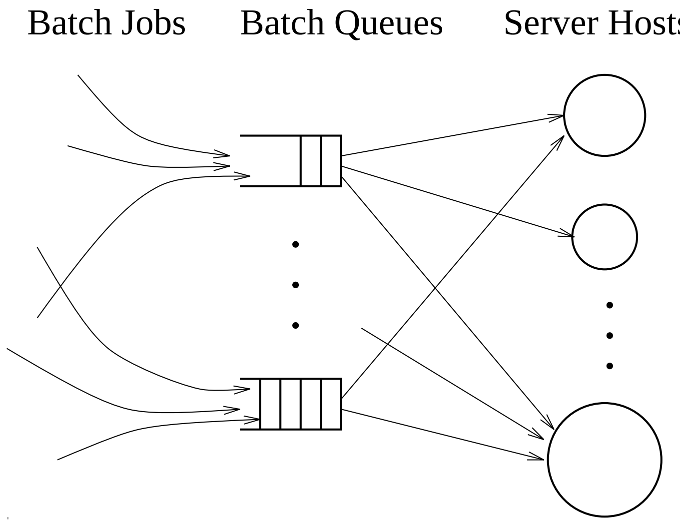
4. Load sharing toolkit: *Lstools*,
   e.g., lseligible, lsplace, rex, lsrun, ch, ...

5. Session jobs: *Lslogin* (Load sharing *rlogin*)

Lsbatch

Hide the distribution complexity, expose the computing power

- Built on top of UTOPIA: share common services
- Cooperating clusters for scalability, autonomy, fault-isolation & management
- Load sharing with real-world constraints
- Delayed dispatch, coordination of mbatchd & sbatchd
- Fault tolerance: no job lost, service available despite host failures
- Dynamic system reconfiguration
- Parallel applications: coordinated placement & control



Lsbatch Configuration

Job queues for classification & customized scheduling:

- Run window
- Scheduling requirements: **`loadSched`**, **`loadStop`** vectors
- Resource usage limits
- Queue priority
- Job nice value
- Host & user groups
- Exclusive execution

Job submission options:

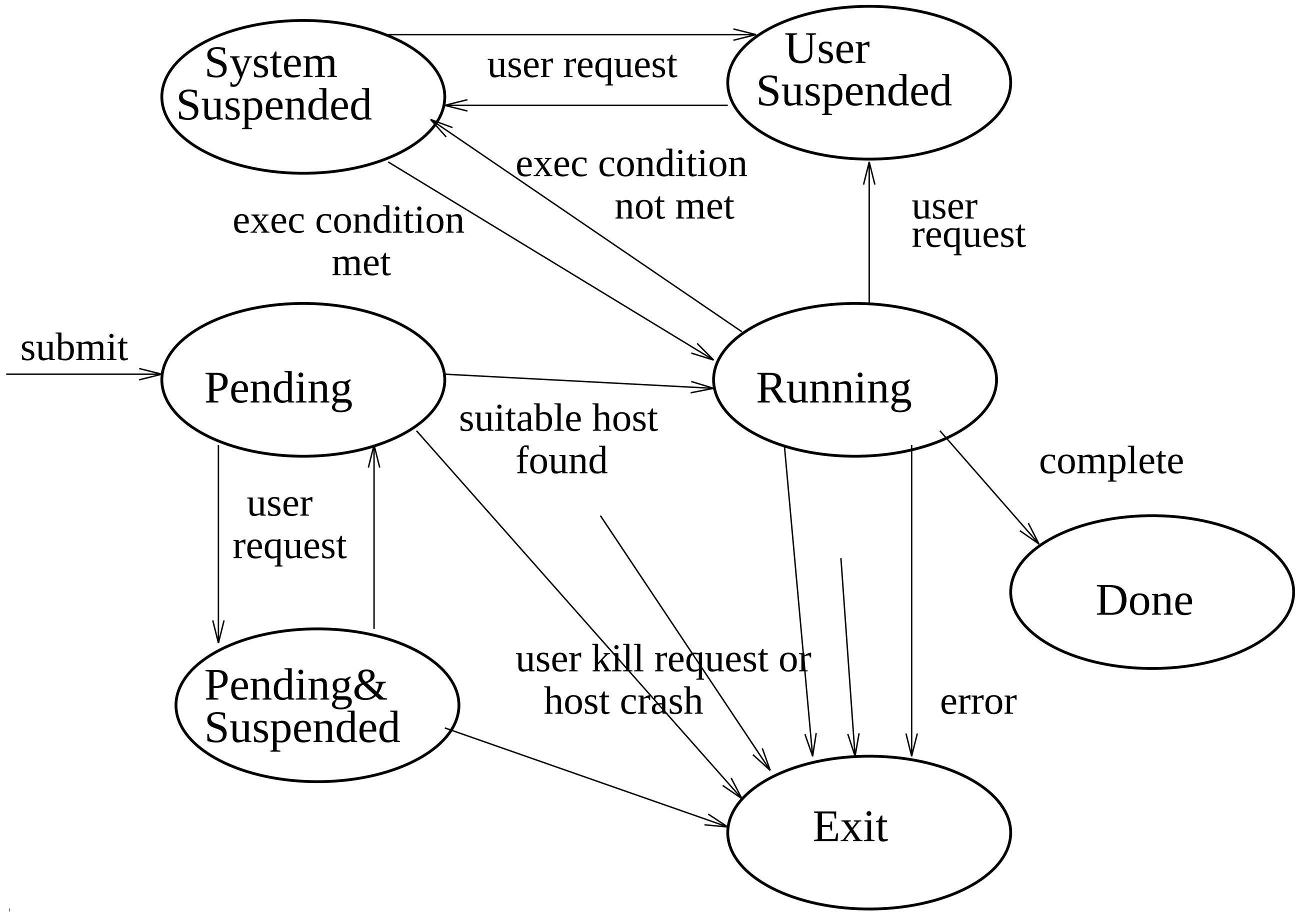
- Number of processors to use
- Job queue to join
- Resource requirements
- Start & termination times
- Restart & checkpoint
- Resource consumption limits
- Exclusive execution

Lsbatch Commands

Power through command options, simplicity with defaults

- job submission: **bsub**
  number of processors; job queue; start & term time; resource requirements & exclusive execution; I/O redirection; restart and job transfer; forced execution on specific host(s)
- job status checking: **bjobs, bjpeek, bqueues, bhosts**
- job and queue control: **bstop, bresume, bkill**
- queue & system management **bqc, breconfig**
- misc: job accounting, quota info, etc

Batch Job State Diagram



Parallel Computing on Workstations

Great promises and challenges

- Equipment already there, but under-utilized
- Very large scale (1000s) with rich resources
- Good support for sequential applications, but not parallel
- Heterogeneous system base
- Not dedicated to parallel applications
- Long communication delay, and bandwidth limitations

Technology Fusion:
.   Parallel Computing and Load Sharing

---

* **Parallel Computing**: *Intra-application* support

e.g., PVM, P4, TCGMSG, DSM, Linda, ...

- support communication and synchronization

* **Load sharing**: *Inter-application* support

- System & host configuration
- dynamic resource load information, host selection & control
- sophisticated scheduling algorithms for multiprogrammed execution: parallel and sequential, interactive and batch
- low cost remote execution mechanisms

Parallel Application Support in UTOPIA

- Generic interface for all applications, interactive or batch
- Change packages to support load sharing, but not application
- User & system job control, e.g., suspend, resume, and kill
- Fast parallel startup using RES (optional)
- Support PVM, P4, MSGLIB, Linda, ...

Interactive submission:

```
prog [-n numProc] [-R resReq] [other arguments]
```

Batch submission:

```
bsub [-n numProc] [-R resReq] [other bsub options]
              prog [arguments]
```

Experience

Working with large complex user sites (1000s of hosts)

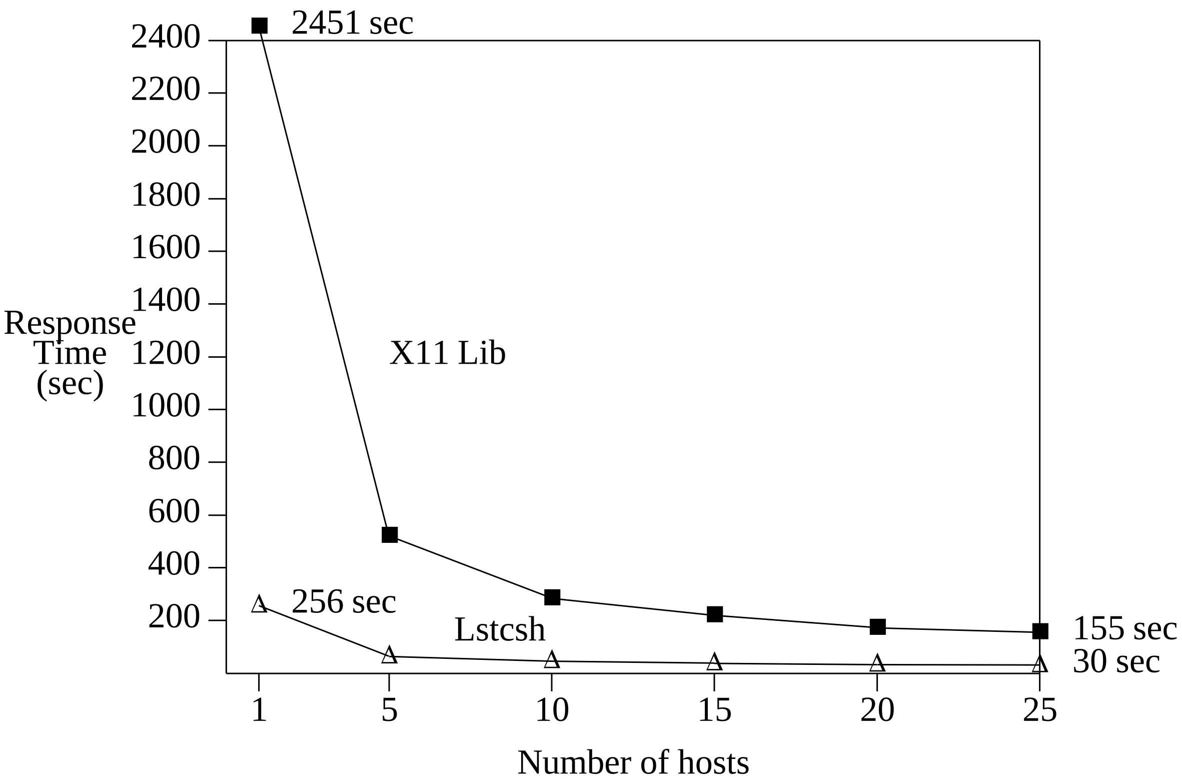
E.g.:

- Electronic device maker: large-scale software/hardware development & ECAD
- Aerospace manufacturer: large-scale numerical computations & MCAD

Impacts of load sharing:

- Reduced, predictable job response time:
  - 20-40% reduction in average response time observed under live workload for sequential jobs
  - Lsmake cuts time to build X11 library from 40 minutes to 2.5 minutes (speedup = 16)
- Controlled access to vast amount of resources for compute-intensive jobs (lsbatch)
- Transparent access to non-local resources (hardware & software)
- Flexibility in system upgrade & management

Lsmake performance



What Are Important in Load Sharing?

- **General purpose**
  - To be effective, must cover & coordinate all/most applications
  - One system for all applications: modular, flexible & extensible
- **Performance is critical**
  - Cost of remote execution & decision shouldn't be a concern
  - Low & controllable network traffic
- **Scalable & heterogeneity support**
  - Greater sharing brings greater benefits
  - Heterogeneity is a fact of life; better off treating it as opportunities
- **Configurability & control**
  - Real systems need knobs & levers
  - To be accepted and used, must be easy for users & keep control for managers

It's all a matter of economics

Concluding Remarks

What does UTOPIA provide?

1. efficient distribution of state info in large-scale systems
2. adaptive task placement advice
3. transparent, efficient remote execution on heterogeneous hosts
4. flexible applications support

Current Status

- Transferred from university to vendor
- Supported on SunOS, IBM/AIX, HP-UX, and Ultrix
- R&D continues ...

(Paper on UTOPIA to appear in *Software – Practice and Experience*; advance copy available via anonymous ftp from csri.toronto.edu, directory csri-technical-reports/257, file 257.ps[.Z])