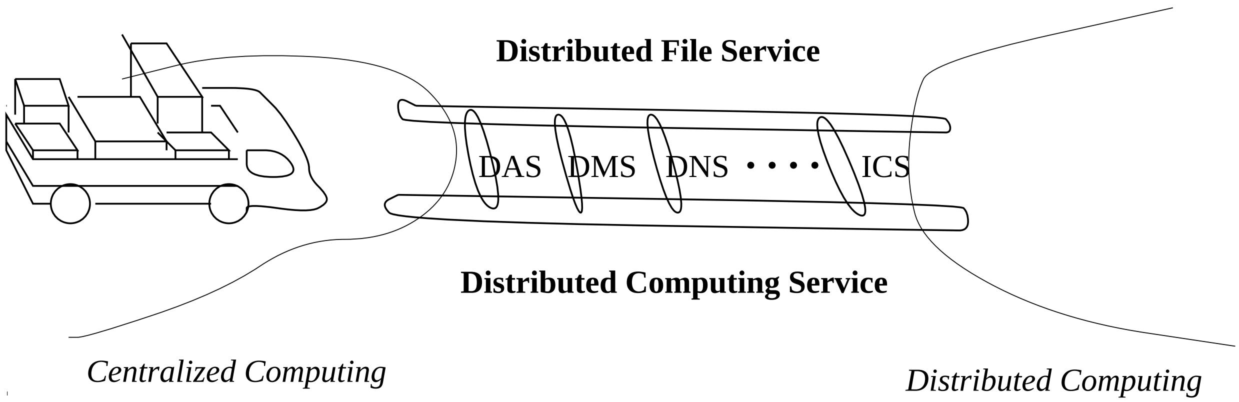
LSF: Load Sharing in Large Heterogeneous Distributed Systems

Songnian Zhou

Platform Computing Corporation
203 College Street, Suite 201
Toronto, Ontario, M5T 1P9 Canada
.
(416) 978-0458
.
zhou@platform.com

* An earlier research version of LSF from Univ. of Toronto is named UTOPIA

Computing in Transformation



Distributed Computing Resource Sharing

Load Sharing: Redistributing system workload among hosts

Motivations:

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

LSF: Distributed Computing System

Integrated framework for distributed resource sharing by all applications through utilities supported by API.

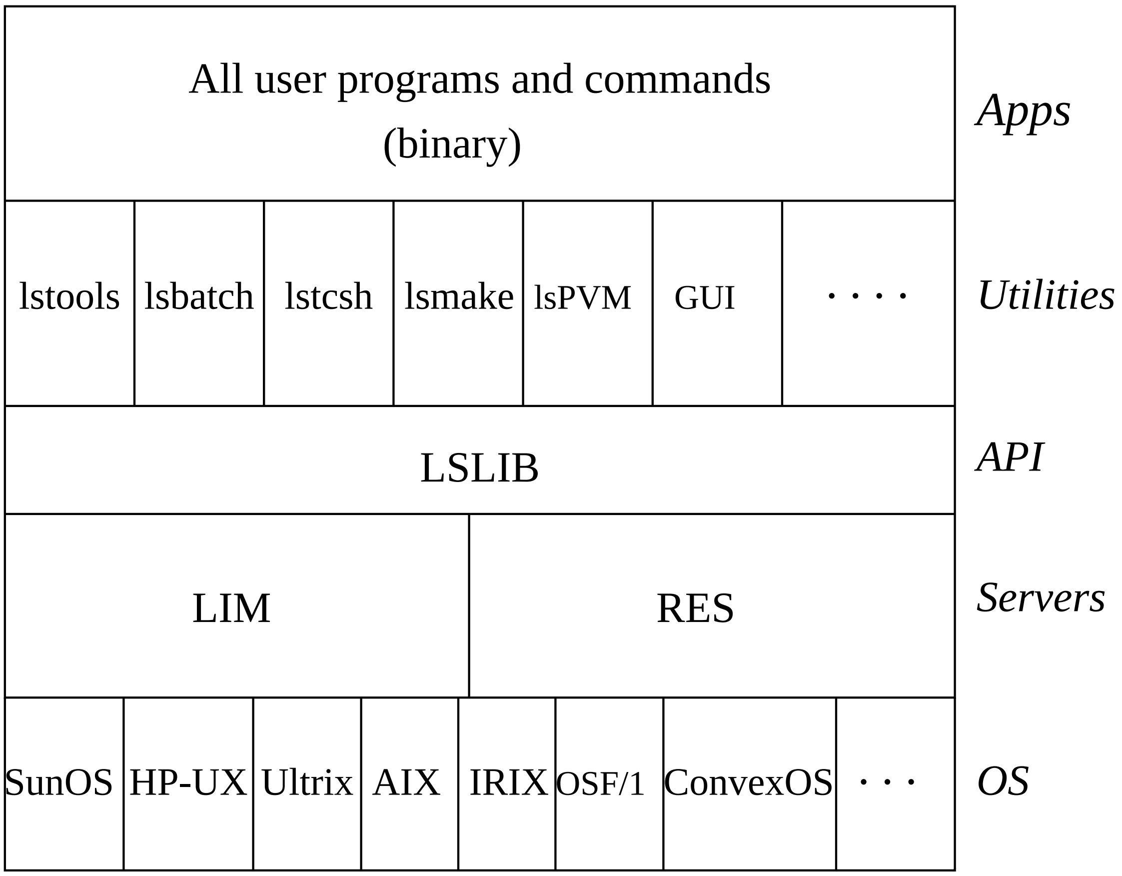
- no limitation to applications: no relinking, any type (sequential or parallel, interactive or batch)
- transparent load sharing among heterogeneous systems
- scalable for enterprise computing
- support autonomy, protection, and configurability
- open system: resource sharing API for app development/porting

LSF Design

Assumptions:

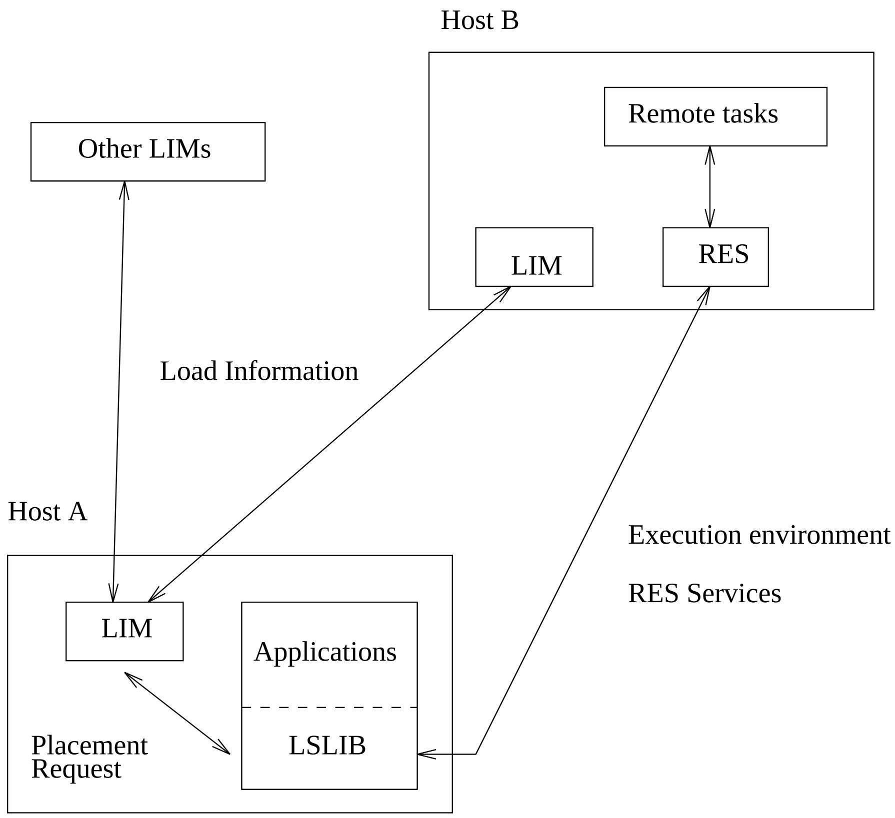
1. remote execution at task initiation time
2. shared file & user space, with relaxations (e.g., NFS automount)

System architecture:



LSF Design

Typical interactions:



Separate scehduling policies
. from remote exec mechanisms
. from API
. from applications

Load Information

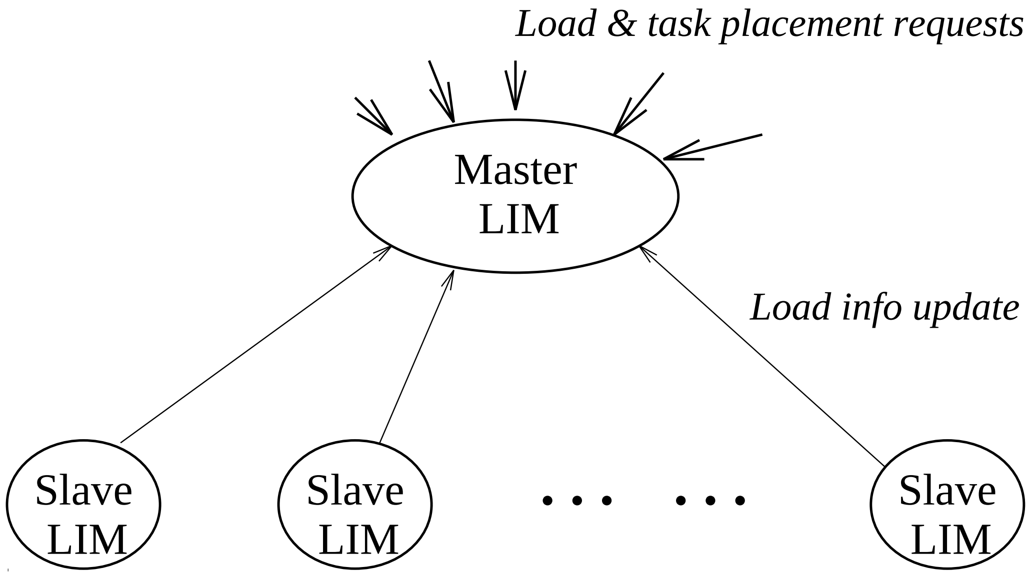
- Average (scaled) CPU queue length & utilization
- Available user memory
- Average paging & disk block transfer rate
- Number of login sessions
- Host idle time
- Available swap space
- Available /tmp space
- Host status: **`ok, busy, unavail, locked,`**
  . **`send/recv only, private, ...`**
- Network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

Used by LIM together with system configuration information

Load Information Exchange & Task Placement

Load sharing cluster: an arbitrary collection of computers that are "close" together & share load extensively (e.g., a department, workgroup, or LAN)

Centralized algorithm:



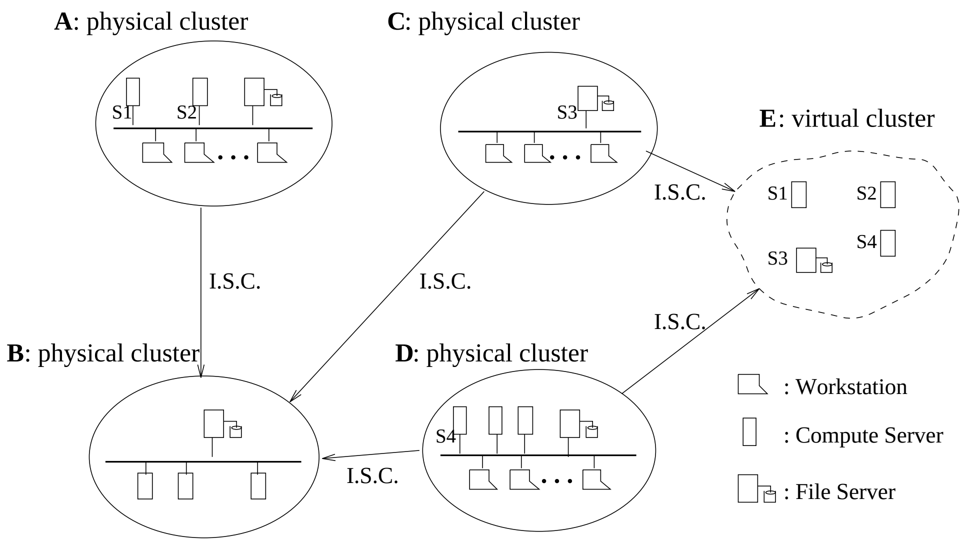
(Think of the master LIM as the traffic cop, or match maker, or benevolent dictator)

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 jobs, 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

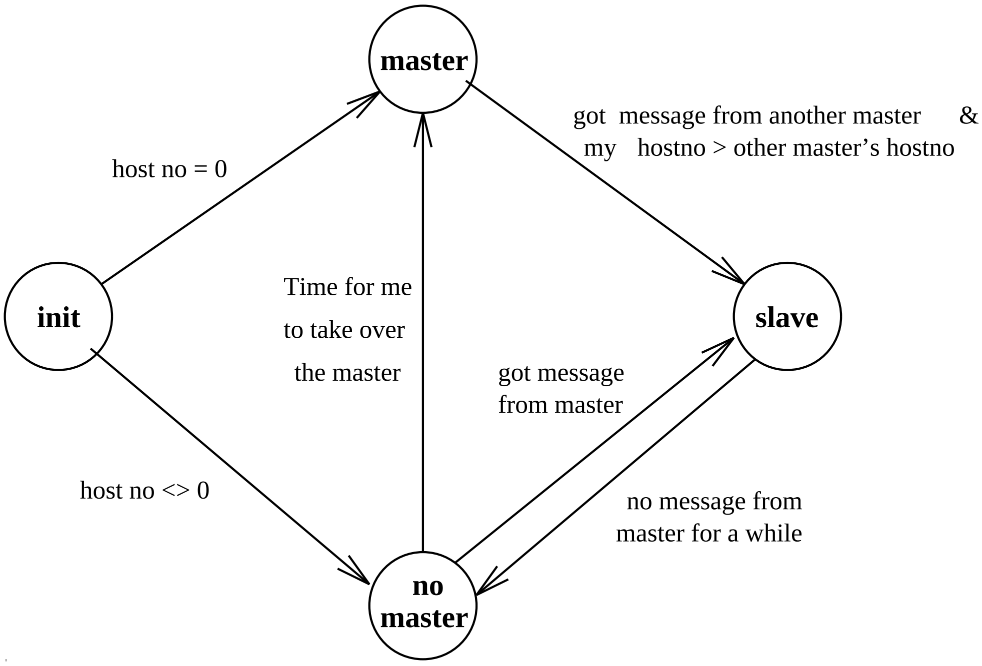
`-R mem:login`

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

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

Master LIM Election & Locating

Master LIM locating:



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

LIM Performance

1. Task Placement Delay:

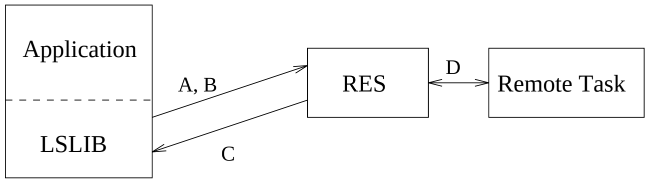
Number of candidate hosts	Number of hosts requested		
	1 host	5 hosts	10 hosts
15	3.8 ms	4.6 ms	5.0 ms
33	4.5 ms	7.7 ms	8.4 ms
60	5.8 ms	14.2 ms	18.5 ms

2. Master LIM CPU & Net Overhead:

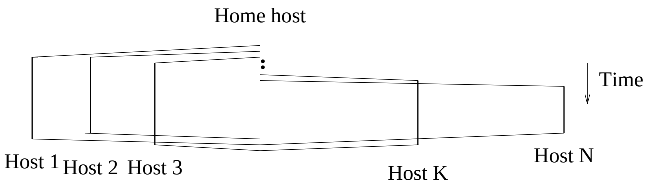
Load sharing cluster configuration	Master LIM CPU (% time)	Network traffic (KB/second)
15-host cluster	0.08%	0.04
33-host cluster	0.15%	0.09
60-host cluster	0.29%	0.17
Two (33, 27) clusters sharing each other	0.22%	0.37
20 50-host clusters 5-way sharing (emulation)	1.1%	1.2

(Measured on an 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

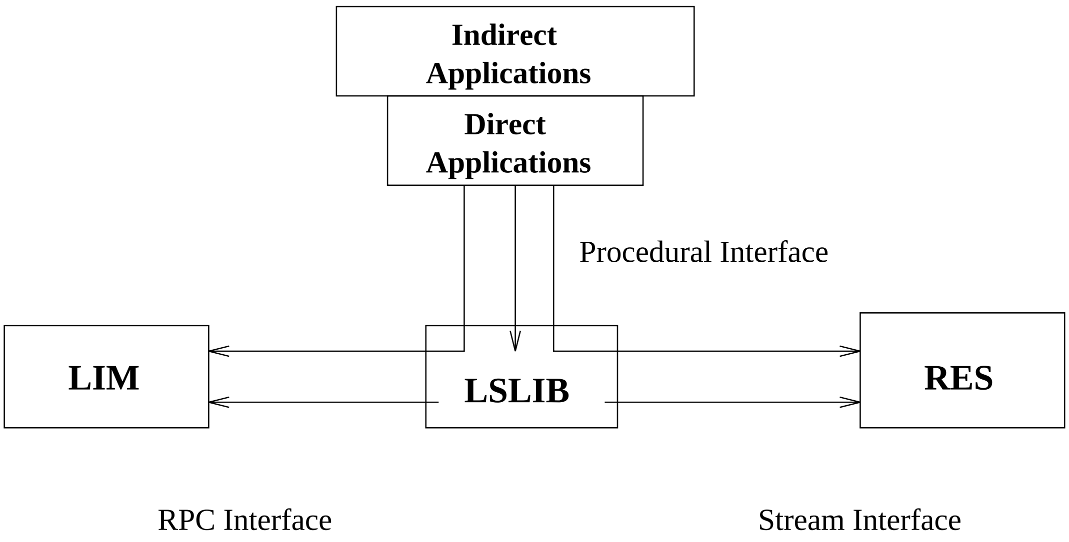
Multilingual applications: dynamic protocol selection

Applications Support

Two classes of applications

- *direct applications*: trucks and trains
- *indirect applications*: “free-loaders”

Load Sharing LIBrary (LSLIB): focal point of LSF



- Load Information Request
- Placement Request
- Configuration Management

- Connection Request
- Remote Execution Request
- I/O and Signal Processing

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;
      }
      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(desthost, 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***):**

	Sequential	Parallel
Interactive	Yes	Yes
Batch	Yes	Yes

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: *lsPVM*, *lsmake*, . . .

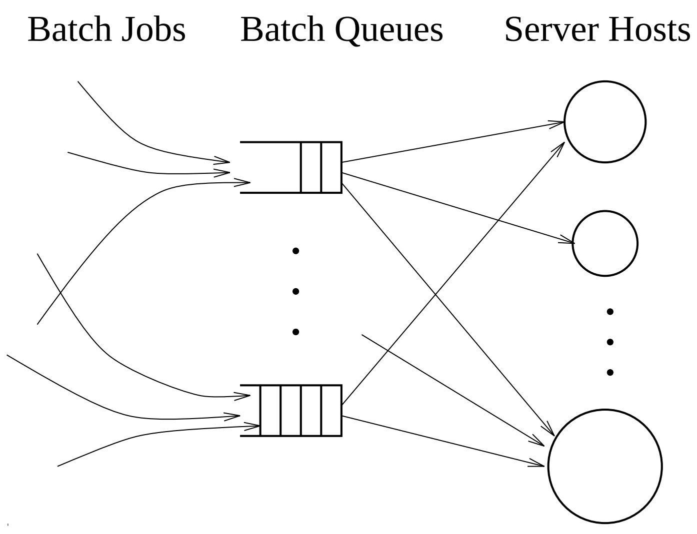
4. Load sharing toolkit: *lstools,*
   e.g., lseligible, lsplace, rex, lsrun, lsgrun, ch, ...

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

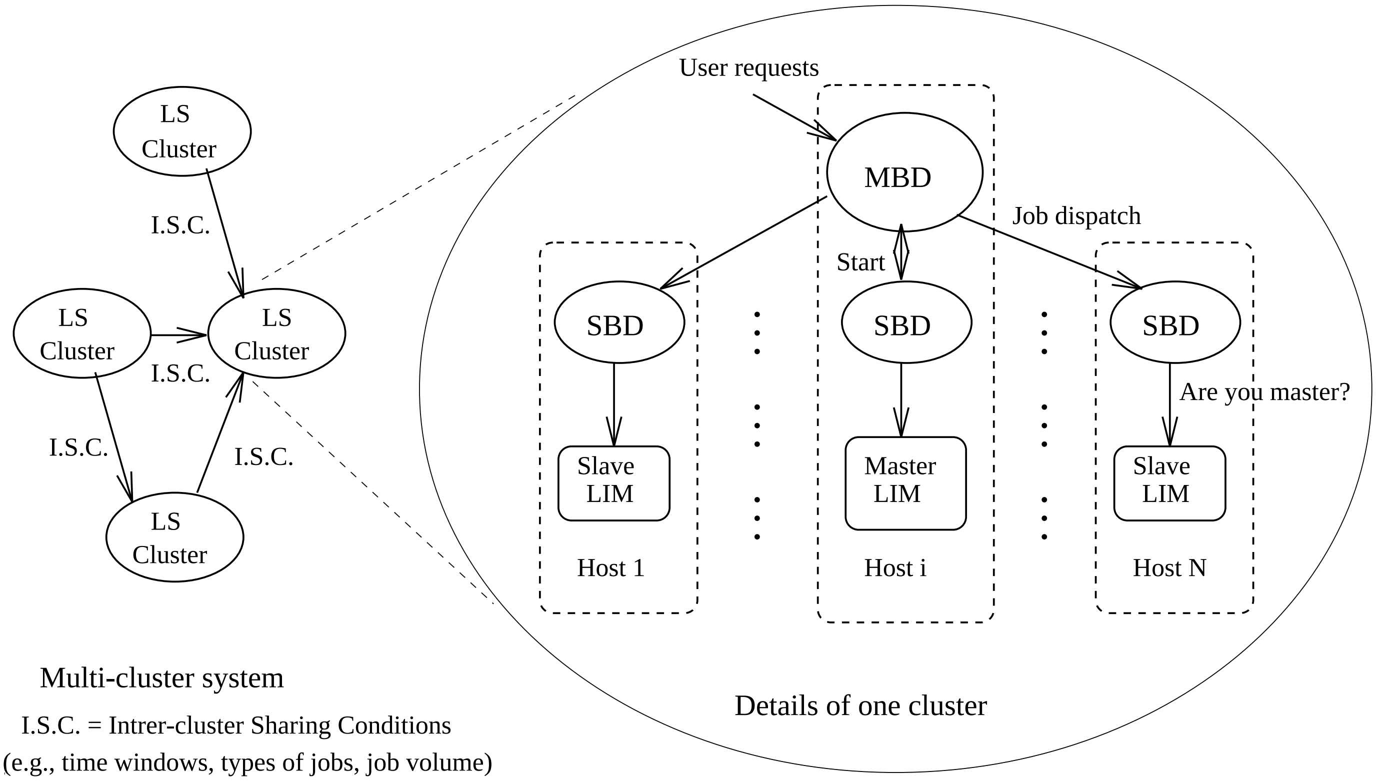
lsbatch

Hide the distribution complexity, expose the computing power

- Built on top of LSLIB: 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
- Detailed job traces & analysis tools
- API & script support for extensibility



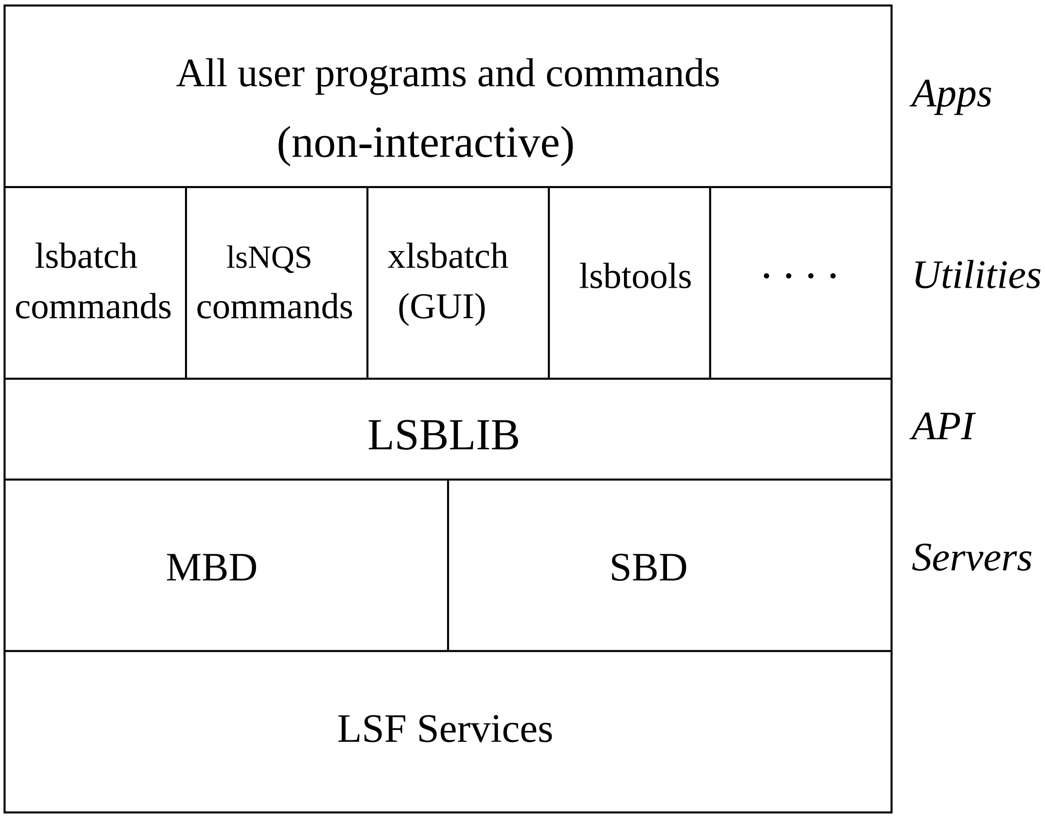
lsbatch Architecture



Multi-cluster system

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

lsbatch Architecture



lsbatch Configuration

Job queues for classification & customized scheduling:

- Run windows
- 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

Example Queues

1. *Priority* Queue:

```
priority=10; nice=10; currently enabled and active;
cpulimit=3600; exclusive jobs not allowed;
run window:   5:18:00-1:7:30  19:00-7:30

           R15S   R1M   R15M   PG       LS    IT  IO
loadSched  20.00  20.00 20.00  1000.0   40    -1  2000
loadStop   30.00  30.00 30.00  2000.0   80    -1  4000

users:     paul, john, bigshot
hosts:     all hosts
```

2. *Idle* Queue:

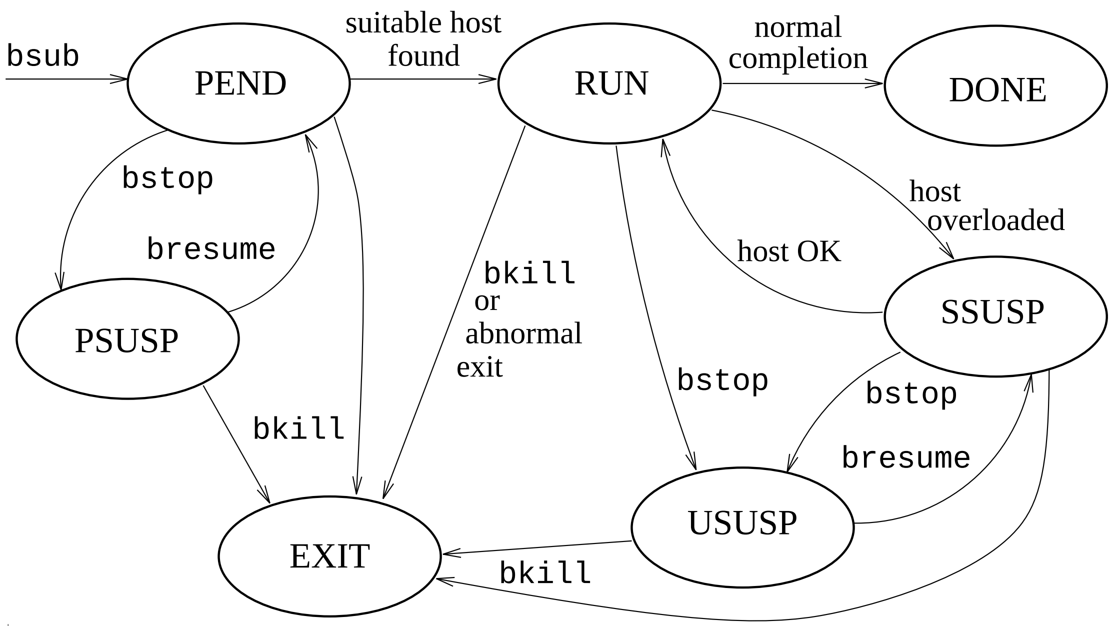
```
priority=100; nice=25; currently enabled and active;
cpulimit=userlimit; exclusive jobs not allowed;
run windows:    always open

             R15S   R1M   R15M    PG     LS    IT  IO
loadSched    0.60   0.30  0.30    4.0    40    10  2000
loadStop     1.50   1.50  1.50    8.0    80     1  4000

users:     all users
hosts:     all hosts
```

•

lsbatch Commands:

- **bsub**
- **bjobs, bpeek, bhist, bstop, bresume, bkill**
- **bqueues, bhosts, busers, bacct, bstat**
- **bqc, breconfig**

Batch Job State Diagram:



Sharing Control

Load sharing desirable, but control necessary in real world
Site sets poilcies, LSF provides mechanisms

- **Time** constraints:
  - Queue configuration (run windows, resource limits)
  - Host run windows (for batch and interactive)
  - Job option (start & termination time)
  - Queue control (**bqc**)
- **Host** constraints:
  - All underlying LSF constraints
  - Server hosts, max number of jobs
  - Job queue's server hosts
  - Exclusive jobs
  - Macro Fair Sharing Scehduling
- **Job** constraints:
  - Special resources
  - Limited jobs on hosts
- **User** constraints: User group for queue
- **Resource** constraints: CPU, memory, ...
- **Load** constraints: by queue and host
- **Cluster** constraints: intercluster sharing conditions

Macro Fair Share Scheduling

A collection of hosts treated as a single resource shared by several (groups of) users according to their shares:

- If no batch job queued up, use as much as you want (FCFS)
- If jobs waiting, give priority to users who use less than their fair share in recent past

Fair Share considered only at job dispatch time.

```
% Split resources according to purchase contributions
Name:   server_pool
Hosts:  big1.engine big2.engine
Shares: (turbine 5) (structure 2) (thermo 1) (others 2)


% Allow other department to share our machines,
% but we have absolute priority
Name:   owner_shared
Hosts:  ws1.mydept ws2.mydept ws3.mydept
Shares: (mydept_members 1000) (others 1)
```

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
- Clusters as scalable compute servers

Technology Fusion:
. Parallel Computing and Load Sharing

---

* **Parallel Computing (languages)**: *Intra-application* support

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

- support communication and synchronization

* **Load sharing (net OS)**: *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 LSF

- 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:

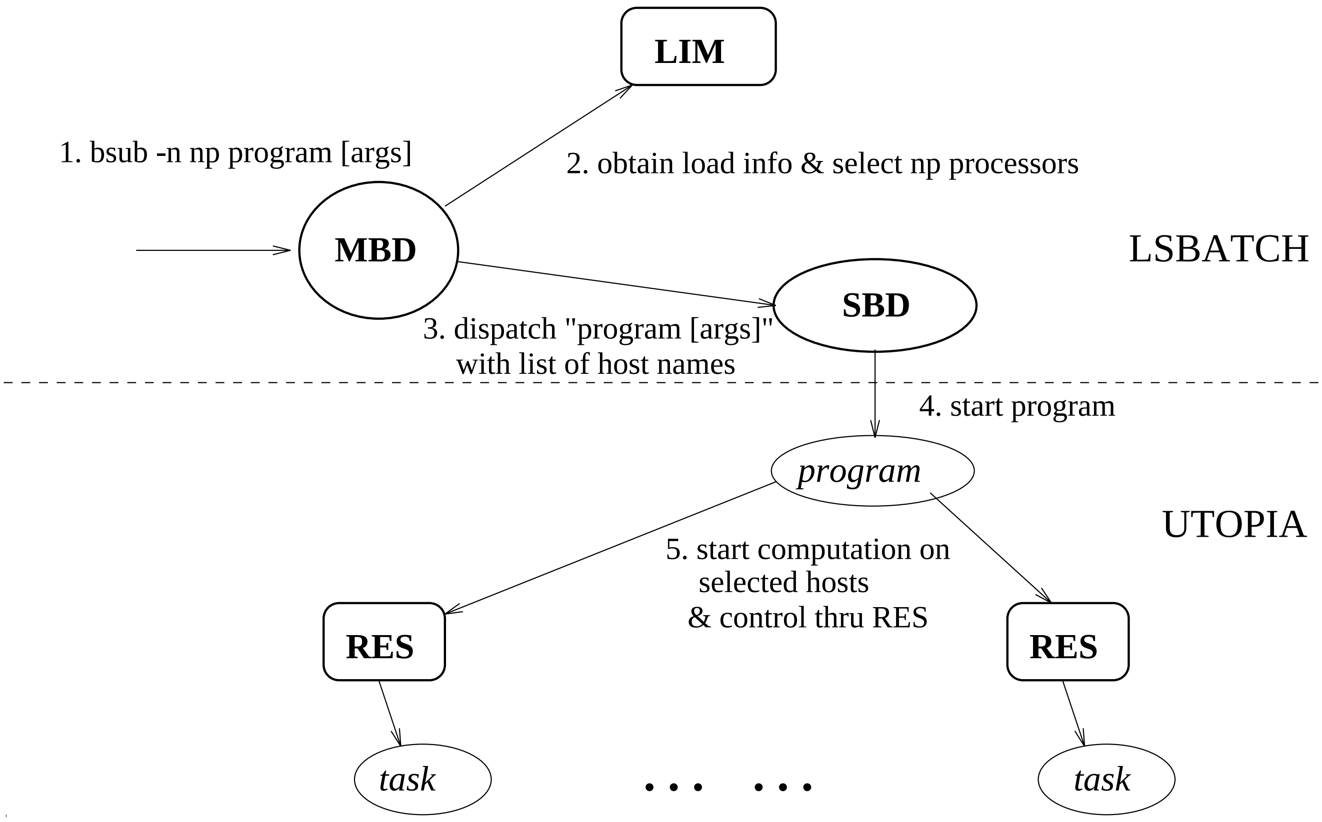
```
prog [-n numProc] [-R resReq] [other arguments]
```

Batch submission:

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

Or: "wrappers" available for generic parallel packages

Parallel Application Support in LSF



Multi-Level Support

- Level 1: allocate processors given resource requirements
- Level 2: L1 + start remote initial processes using RES
- Level 3: L2 + application control (via RES or within app)
  E.g., suspend, resume, kill, checkpoint
- Level 4: L3 + start all remote tasks & master-slave communication

// Application Support: Examples

lsPVM:

- pvmds started at beginning of execution & cleaned up at end
- No hostfile: dynamic processor allocation
- Allow multiple PVM jobs of same user on same host
- No change to application code
- Same binary runnable interactively or under lsbatch
- Level 3 support

TCGMSG: daemon-less package — use RES

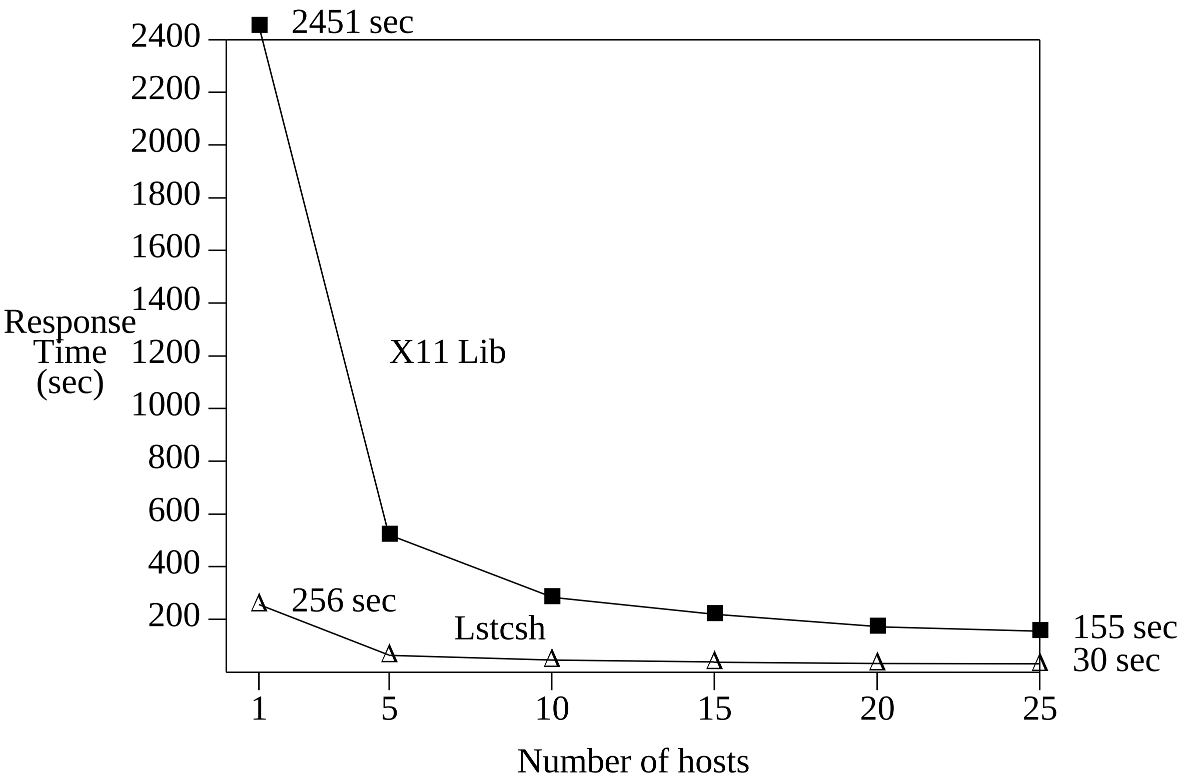
XYZ (propritary package): supported by LSLIB and lsbatch w/o knowledge

lsmake: Level 4 support

HPF compiler: Level 3, or 4 support

lsmake performance

Max Speedup: 16



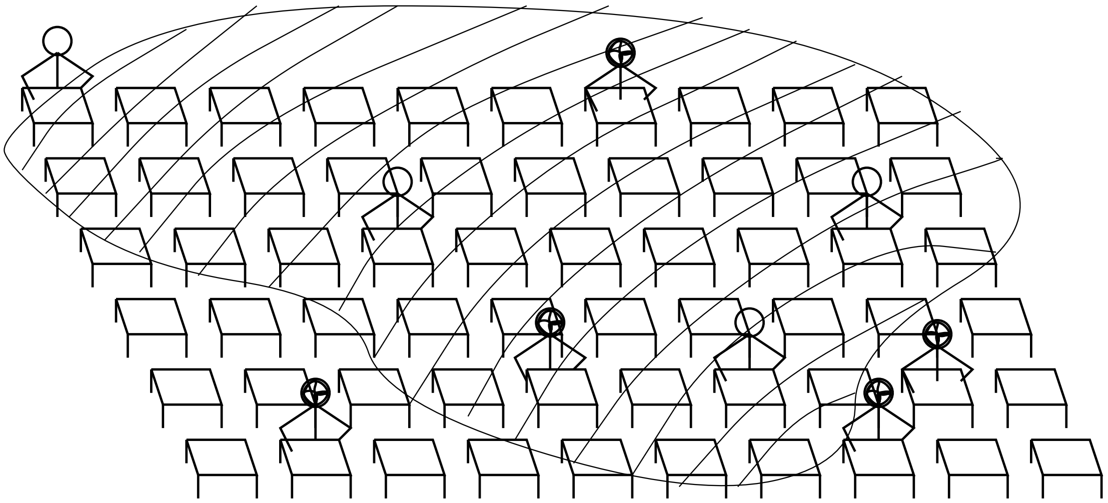
Compared to Sprite OS: Max Speedup: 5

Using a cloud to compute a cloud

Parallel computation of hydrogen cloud on workstations

(with Norman Wilson, RCS & Bill Keogh, Astrophysics, UofT):

- PVM + LSF
- Coresidence with interactive workload w/o interference
- Speedup 56 on 60 hosts: 3 months downto day 16 hours
- Non-vectorizable code: 10 MFLOPS on Cray, 100 MFLOPS on workstations



60 SUN IPCs supporting interactive users

Experience

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

- **Electronics** (e.g., Motorola): hardware development & ECAD
- **Software** (e.g., Bell Northern Research): CASE
- **Aerospace** (e.g., Pratt & Whitney): large-scale numerical computations & MCAD
- **Petroleum** (e.g., Shell Oil): reservoir, seismic computation
- **Financial** (e.g., Canadian Imperial Bank of Commerce): risk analysis, portfolio evaluation
- **Pharmaceutical** (e.g., Glaxo): molecular chemistry
- **Utilities** (e.g., Ontario Hydro): power grid simulation, neuclear waste disposal
- **Scientific & Research** (e.g., MITRE Co.): numerical, simulation, signal processing

LSF *used as company-wide computing infrastructure to tie equipment together for transparent enterprise computing*

Impacts of load sharing

- Reduced, predictable job response time:
  - **30-40% reduction** in average response time over hundreds of workstations observed under live workload
  - 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)
  - **2x** batch job throughput over hundreds of hosts observed under live workload
  - Parallel computation rivals Cray at little additional cost
- Transparent access to non-local resources (hardware & software)
- Flexibility in system upgrade & management

Custom // Applications

- Engine "plumbing" design — N x N intersection checking
  . overnight run reduced to hourly runs

- Grid generation for irregular CFD
  . hour reduced to minutes

- Diaper "crash" test - CFD & mechanical

Methods:

- as shell script (using lstools)
- programming using LSLIB

LSF User Benefits

- Productivity Boost
  - Shorten job response time
    - Run jobs on fast, lightly loaded hosts
    - Parallel job execution
  - Increase batch job throughput
  - Transparent access to resources
  - Shorten product design cycle
- More Efficient Use of Resources
  - Incremental upgrade
  - Server pools
  - Better utilization of software licenses
  - Balance load at login time and for heavy applications
- Better System Planning and Management
  - Identify bottleneck resources
  - Know when and what to upgrade
  - Enable client-server architecture
  - Integrate of heterogeneous systems
  - Ease system administration
- Enable Downsizing from Mainframes and Supercomputers
- Open Environment for Growth and Customized Applications
- Integrated Framework for Distributed Resource Sharing

Case Studies

- BNR/NT
  - 4000 workstations, 7000 engineers
  - "Get rid of mainframes"
  - LSF running on 800 workstations for 2 years
  - "LSF is basis of our corporate vision of distributed computing"
  - So far, mostly interactive; getting into batch
- Pratt & Whitney
  - Corporate-wide deployment
  - Sent back Cray Y/MP
  - 5x computing on workstations at low cost
  - Resource utilization rising
  - Broad mixture of workload: interactive & batch, sequential & parallel
- Western Digital, Motorola, TI, ...
  - "Engineers' productivity critical"
  - Bursty workload forces engineers to become "resource hunters on the network"
  - Computing demands escalating but computer system "out of control" — LSF's job logging useful
  - LSF's batch and parallel support usable "out of box"
  - Integration with CAD tools desirable

Case Studies (cont.)

- Grumman
  - "Mainframe is fine, but can't afford maintenance"
  - Workstations have raw horsepower, but engineers can't deal with 100s of "husky dogs"
  - Success case: replaced one mainframe with 10 workstations for the same workload
- Bank X
  - More and more relying on modeling: investment, currency trading, options, ...
  - 2 types of applications: overnight run, interactive on-line options eval
  - Simple parallelism, massive amount of computation
  - Naive users w/ quick money for more money

Case Studies (cont.)

- Fortune 100 consumer product company
  - Product design using CFD, structural, thermo computations
  - Better products lead to edge over generic brands
  - But, "computing must be cost-effective: supers are not, a cluster of workstations aren't either"
  - Need a single system, need integration, need 1-stop shop!
  - "It's too expensive for us to maintain a large system staff & build all expertise"
  - "LSF does 90% of what we need; remaining 10% is integration with app software"
- University of Rome, Washington, ...
  - Broad mixture of scientific and teaching workload
  - Prefer cluster computing rather than workstations
  - Most "dumb" logins onto one host: *lslogin* useful
  - Distribute heavy jobs w/o impacting interactive users

Platform's Strategy

- *Working with challenging, sophisticated user sites to improve technology and solve real world problems*

- *Working with system vendors: IBM, HP, DEC, SUN, SGI, Convex, Cray Research, SCO, ...*

- *Working with software vendors and system integrators using LSF as distributed applications platform*

Architecture, Architecture, Architecture!

Layered Architecture

- o modular (like lego blocks)
- o add-in, add-on components
  (distribued utilities, applications)
- o "arbitary" combinations
  e.g., lsPVM on lsbatch, lsmake in lstcsh

Distributed Clusters

- o efficiency & locality
- o loose cooperation among
  separate admin domains

Heterogeneity

- o uniform framework
  to avoid "all-pair" problem
- o lowest level, hidden

"Meta Architecture":

("ISO/OSI")

Layers & Modules
Distribution
Heterogeneity

Heterogeneity Support in LSF

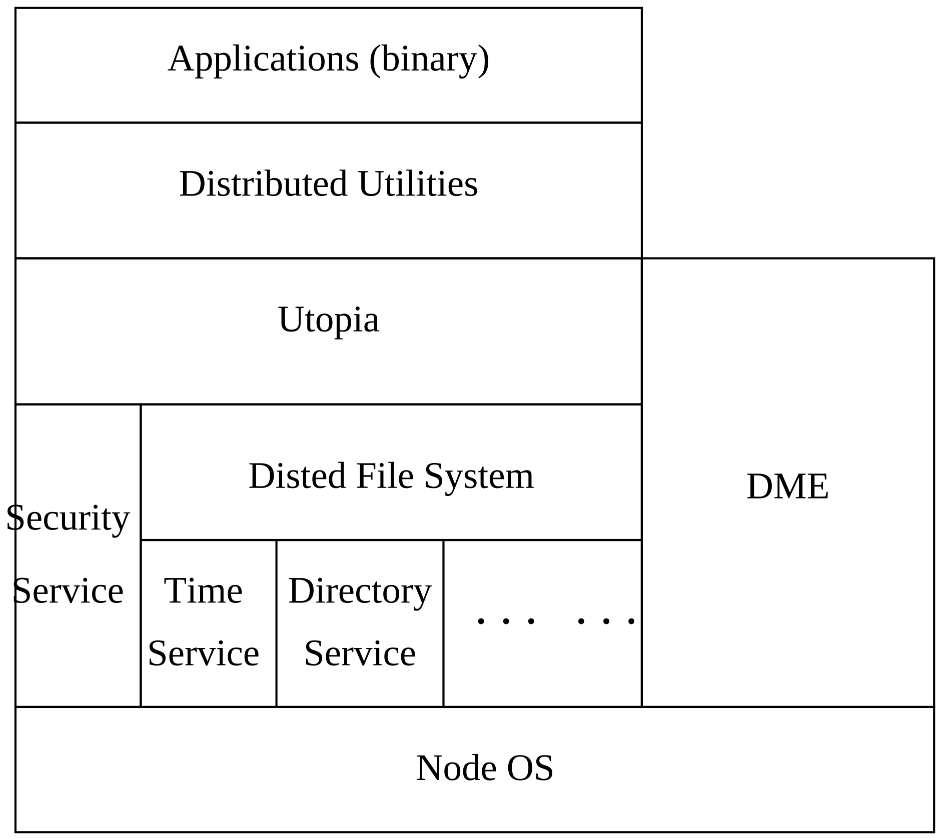
All UNIX platforms interoperate with full transparency

- All communication with XDR
- Special resource requirements in tasks lists
- Uniform support of signals, pseudo tty, etc
- CPU speed scaling, multiprocessor considerations

Open Systems Approach

- Users demand it
- Leveled playing field for all vendors
- Contribution to the technology by all parties
- General, extensible architecture for future growth

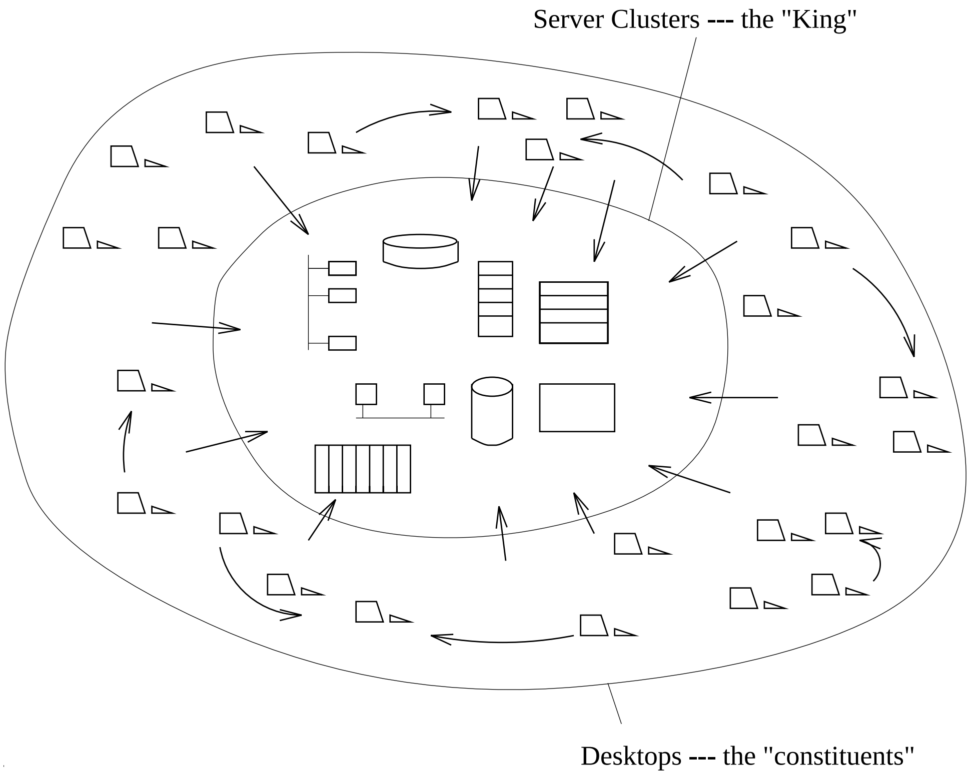
LSF and OSF/DCE, DME



The dynamic resource load and job flow info provided by Utopia
is valuable to DME in system/network management

Emerging Distributed System Structure

- Cheap desktops + powerful server cluster more cost-effective than workstation + fileservers
- Single-system view is necessary for the server cluster
- The transparent flow of jobs into the server cluster is required



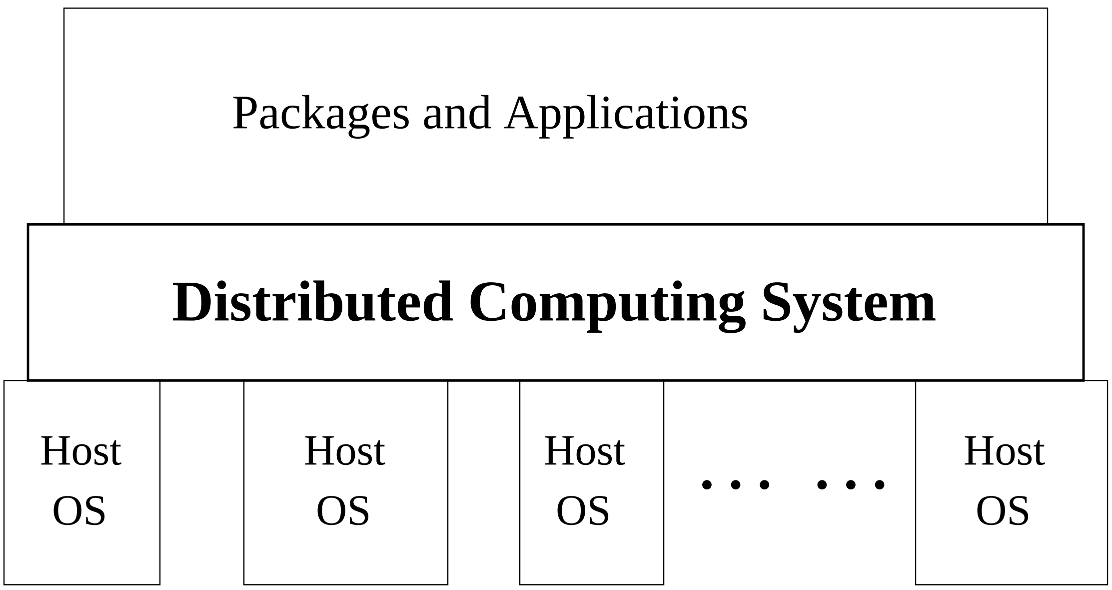
Conslusion

Glue is needed to integrate pieces together seemlessly to get "best of both worlds"

What should the glue look like?

- supports all applications, unchanged
- integrates heterogeneous systems
- has a general-purpose API - open system
- has enough nobs and levers for control

A distributed computing system on top of host OSes



LSF Availability

- Distributed by Platform
- Bundled with Meta Series, HP Clusters from Convex
- Bundled with ALPHA AXP Farm from DEC
- Distributed & supported worldwide by Convex, DEC, ...

Documents:

Technical papers available via anonymous ftp from

`ferut.sys.toronto.edu (128.100.2.220):`

LSF (formally UTOPIA) design paper:
`white-technical-reports/257/257.ps.Z`

lsbatch design paper:
`white-technical-reports/286/286.ps.Z`

LSF Product information: info@platform.com