IBM Platform LSF Overview & Directions

IBM Platform LSF Family

IBM Platform LSF: The HPC Workload Management Standard

- Advanced, feature-rich workload scheduling
- Robust set of add-on features
- Integrated application support

Most Complete

- Policy & resource-aware scheduling
- Resource consolidation for max performance
- Advanced self-management

Most Powerful

- Thousands of concurrent users & jobs
- Virtualized pool of shared resources
- Flexible control, multiple policies

Most Scalable

- Optimal utilization, less infrastructure cost
- Better productivity, faster time to result
- Robust capabilities, administrative productivity

Best TCO

Advanced Features

- Heterogeneous platform support
- Supports traditional as well as private & public cloud infrastructures
- Comprehensive, intelligent scheduling policies
- GPU-aware scheduling
- Advanced self-management
- Extensible security
- Always-on availability
- Delegated administration
- Resizable jobs
- Live cluster re-configuration
- Complete API, web-services interfaces
- Ease-of use features
- much more …

Intelligent, Policy-driven Scheduling Features

- Fairshare scheduling
- Topology & core-aware scheduling
- Preemption
- Backfill scheduling
- Resource reservations
- Serial/Parallel controls
- Advanced reservation
- Job starvation
- License scheduling
- SLA based scheduling
- Absolute priority scheduling
- Checkpoint / resume
- Job arrays
- GPU-aware scheduling
- Plug-in schedulers

Advanced scheduling features are the key to maximizing efficiency and productivity while minimizing cost.

Product Functionality Scales to Meet Your Evolving Requirements

Express Edition

Entry level product for those with simple scheduling requirements and small clusters

Standard Edition

Full featured version supporting a wide range of scheduling policies and job management functions.

Advanced Edition

Architecture to support extreme scalability and throughput 100k+ cores & concurrent jobs

Fully Certified and Supported

- **Commercially supported by IBM Platform Computing**
  - Two decades of HPC experience
  - Significant investment in certification & validation
  - Professional Services delivers speedy installation and application Integration
  - Training courses available to help you get full understanding and value of the HPC software
- **Support "from the source"**
  - 24x7 global support
  - eSupport
  - Automated subscriptions for bulletins, patches & updates
  - Software upgrade available with paid professional support & maintenance service
- **Hardware and software support from the same source**
  - IBM – the Technical Computing

"I really don't know how we would have done it without Platform and the support that we get."
Dr. Athanasoulis, Harvard Medical School

"This was a great support experience for me. One of the best I have had. Even better considering the fact we are in different parts of the world."
Major Asian Energy Corporation

"Thanks a lot. You saved me. All my budget for Platform Support over the past few years has yielded a return as good as gold."
Prestigious US School of Public Health

IBM Platform LSF Objectives

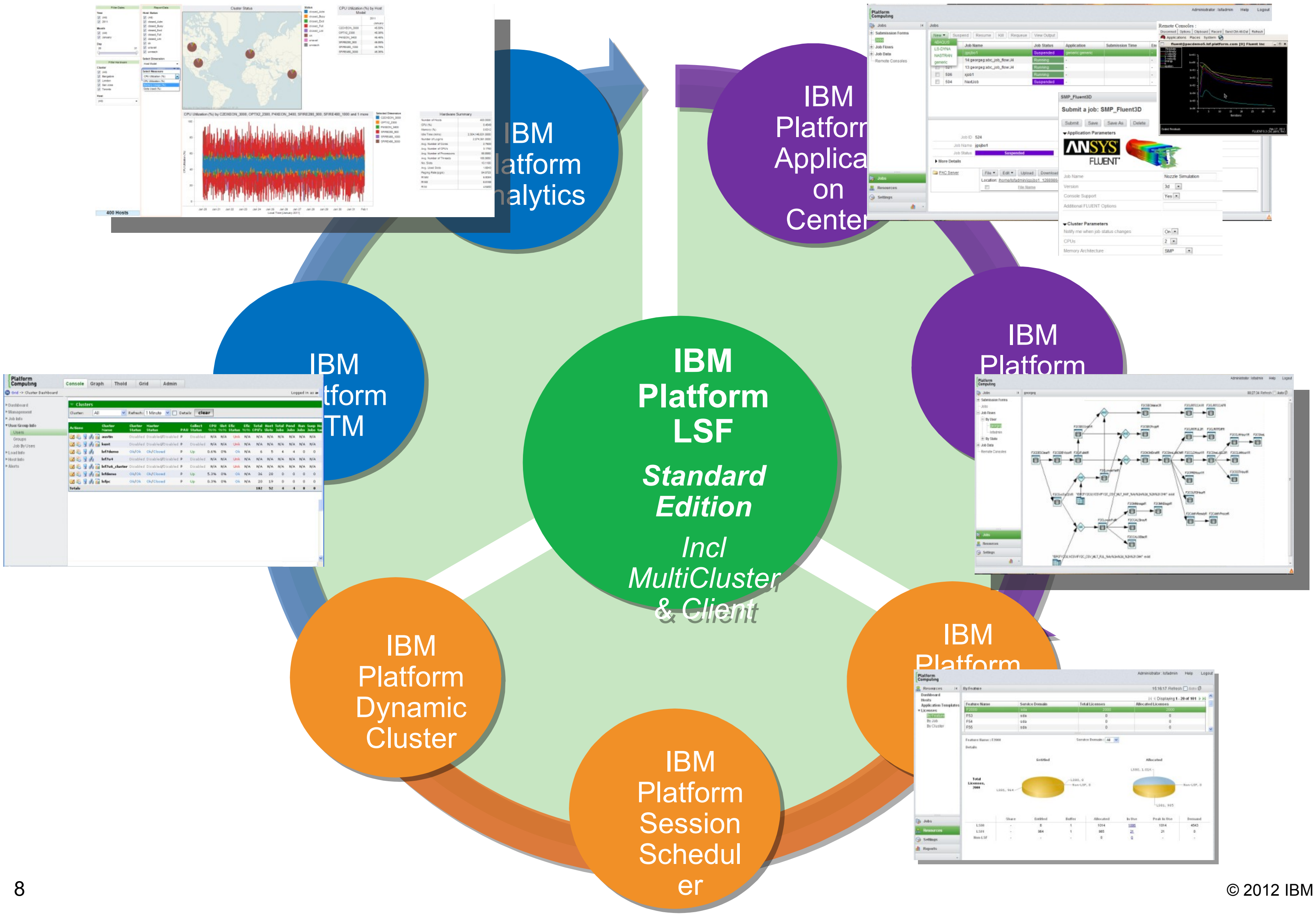
A Simple Idea:

- Virtualize a heterogeneous infrastructure
- Optimally manage workloads according to policy

Key benefits:

  - improves throughput - > competitive advantage
  - makes computers *work harder* - > financial advantage

IBM Platform LSF Product Family



Platform LSF 9.1 Overview

- Agenda
  - Platform LSF 6, 7, 8, 9.1
  - Platform LSF 8, 9.1 – Key Features
  - Platform LSF 9.1.1 Enhancements

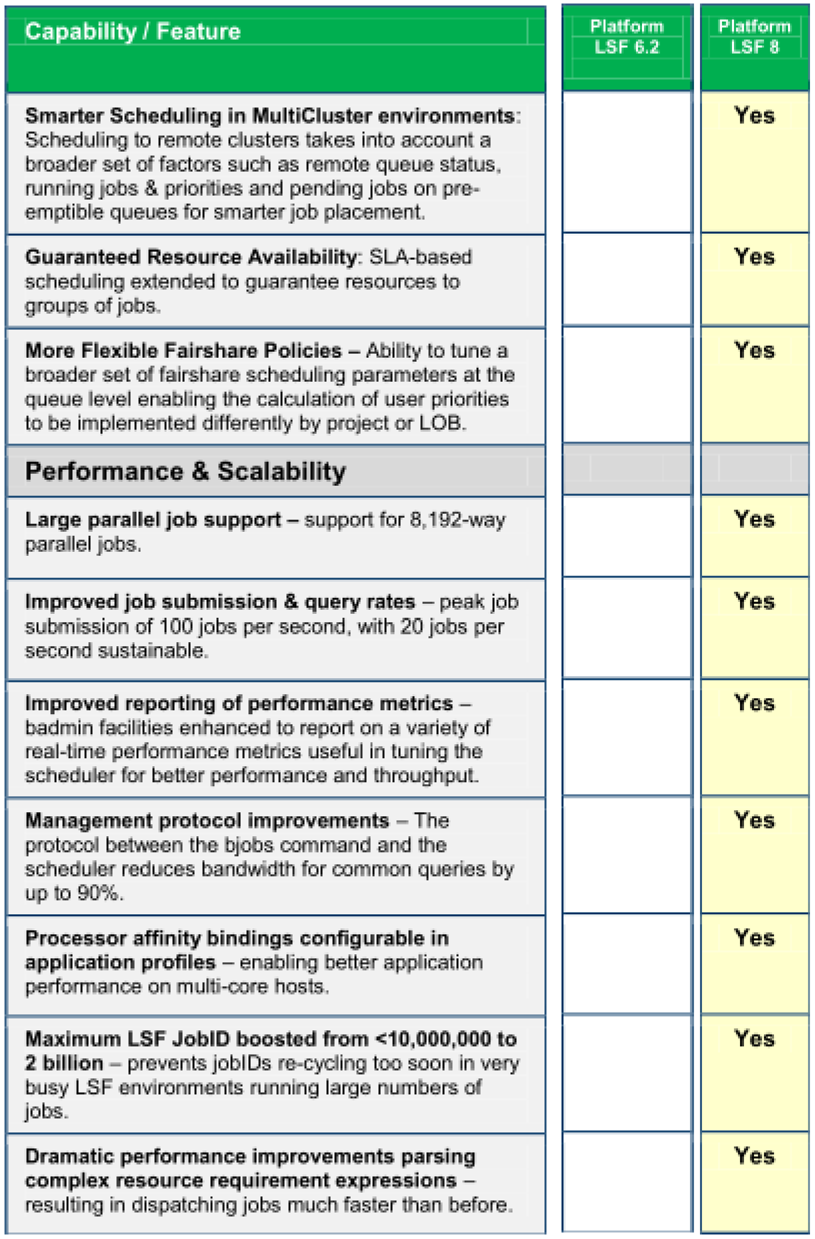
LSF Feature Matrix

Capability / Feature	Platform LSF 6.2	Platform LSF 8
Approximate release date:	Nov 2005	Dec 2010
Scheduling Enhancements		
Exclusive host support – allows only jobs that require an explicit resource to run on hosts having the resource.	Yes	Yes
Improved resource granularity controls – reserve numeric resources based on slots, jobs or hosts.	Yes	Yes
Interruptible backfill – allows reserved job slots to be used by low priority small jobs that will be terminated on start of large parallel jobs.	Yes	Yes
Ability to express multiple first execution host candidates – provide a list of eligible candidate hosts at the queue level or during job submission.		Yes
EGO-enabled SLA scheduling – enables additional scheduling flexibility with SLA-based scheduling in EGO environments. LSF service classes can be associated with EGO consumers.		Yes
New blaunch framework – extends the existing PAM/Taskstarter framework providing a simpler and more flexible mechanism for launching parallel jobs.		Yes
Absolute priority scheduling – Enhances LSF's prior facility to allow job priorities to gradually "creep" over time to avoid resource starvation scenarios by allowing jobs priorities to eventually exceed queue priorities ensuring that jobs don't languish indefinitely.		Yes
Smarter exception handling in "black hole" algorithms – the job EXIT_RATE, beyond which a host is considered to be a "black hole" host, is enhanced to take into account that multi-core hosts have more capacity by scaling threshold exits rates automatically.		Yes
Exclusive sharing policy enhancements – enables pre-emptive scheduling policies to support both backfilled and exclusive jobs ensuring that critical jobs run regardless of other workload policies in use.		Yes

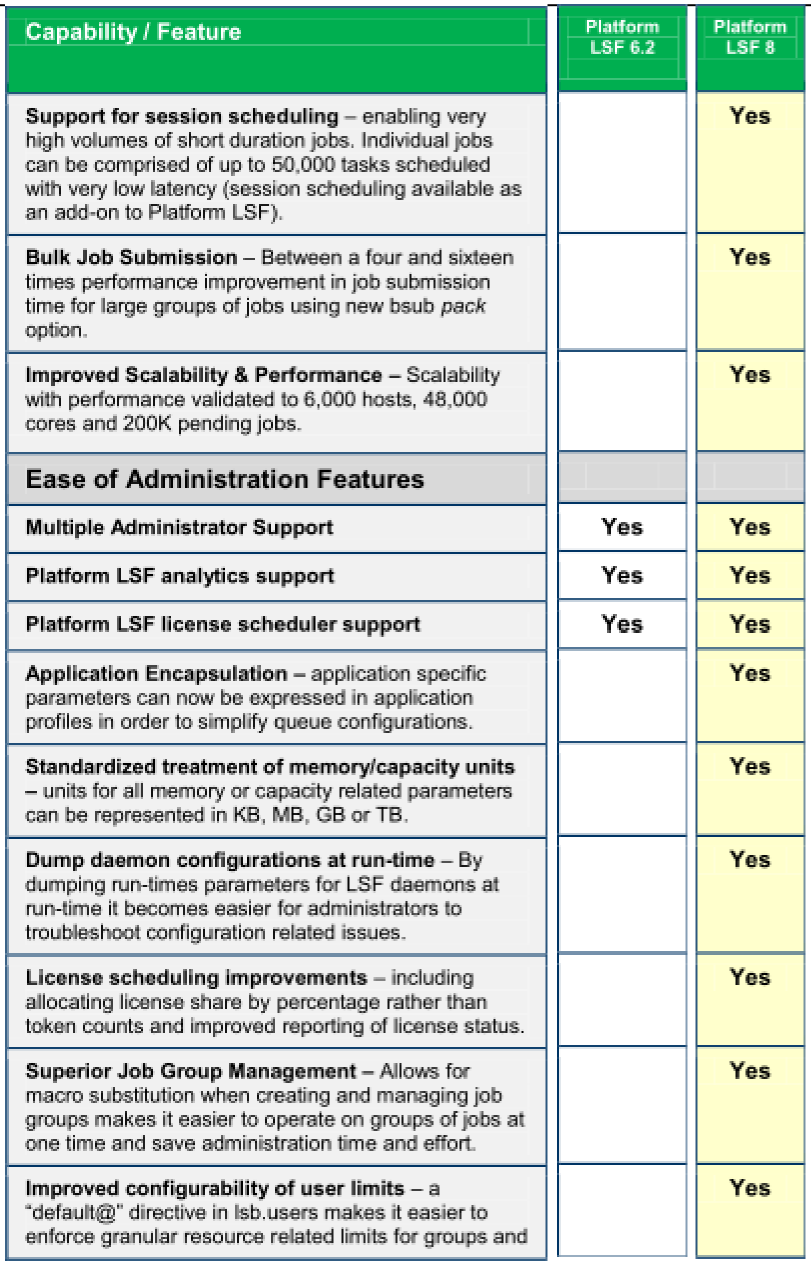
Capability / Feature	Platform LSF 6.2	Platform LSF 8
Job pre-emption based on resource requirements – improved scheduling functionality allows jobs to be pre-empted based on resource requirements of a more business critical job.		Yes
New Advanced Reservation Functionality – enables granular controls and modifications to parameters associated with future reservations.		Yes
Dynamically change run-time estimates – By enabling run-time estimates to be revised "on the fly", and by normalizing estimates to account for relative performance differences of hosts, the scheduler can make more effective decisions and maximize performance and utilization.		Yes
Processor affinity for parallel jobs – enables workloads to be bound to particular CPUs and cores with configurable "packing" strategies to maximize cache efficiency, performance and utilization.		Yes
Generalized resource reservations – resources can be reserved for pending resources in advance ensuring that resources are allocated to pending jobs in job priority order.		Yes
Dynamic priority adjustments – user priorities calculated for fairshare allocation now allows administrators to apply different "weighting factors" to terms that define scheduling priority in order to alter fairshare scheduling to their own requirements.		Yes
Compound resource requirements – Allows different resource requirements to apply to different job slots within a single job, useful for distributed programs where different threads may have different underlying resource requirements.		Yes
Compute units – allows scheduling based on structures that mimic a network architecture or rack structure to enable smart topology aware scheduling.		Yes
Resource Reservation Limit Enhancements – RESRSV_LIMIT expression allows a resource requirement "range" to be expressed at the queue so that resource requirements can be expressed on a per-job or application basis to avoid resource requirements being over-estimated.		Yes

10

LSF Feature Matrix



Capability / Feature	Platform LSF 6.2	Platform LSF 8
Smarter Scheduling in MultiCluster environments: Scheduling to remote clusters takes into account a broader set of factors such as remote queue status, running jobs & priorities and pending jobs on pre-emptible queues for smarter job placement.		Yes
Guaranteed Resource Availability: SLA-based scheduling extended to guarantee resources to groups of jobs.		Yes
More Flexible Fairshare Policies – Ability to tune a broader set of fairshare scheduling parameters at the queue level enabling the calculation of user priorities to be implemented differently by project or LOB.		Yes
Performance & Scalability		
Large parallel job support – support for 8,192-way parallel jobs.		Yes
Improved job submission & query rates – peak job submission of 100 jobs per second, with 20 jobs per second sustainable.		Yes
Improved reporting of performance metrics – badmin facilities enhanced to report on a variety of real-time performance metrics useful in tuning the scheduler for better performance and throughput.		Yes
Management protocol improvements – The protocol between the bjobs command and the scheduler reduces bandwidth for common queries by up to 90%.		Yes
Processor affinity bindings configurable in application profiles – enabling better application performance on multi-core hosts.		Yes
Maximum LSF JobID boosted from <10,000,000 to 2 billion – prevents jobIDs re-cycling too soon in very busy LSF environments running large numbers of jobs.		Yes
Dramatic performance improvements parsing complex resource requirement expressions – resulting in dispatching jobs much faster than before.		Yes



Capability / Feature	Platform LSF 6.2	Platform LSF 8
Support for session scheduling – enabling very high volumes of short duration jobs. Individual jobs can be comprised of up to 50,000 tasks scheduled with very low latency (session scheduling available as an add-on to Platform LSF).		Yes
Bulk Job Submission – Between a four and sixteen times performance improvement in job submission time for large groups of jobs using new bsub *pack* option.		Yes
Improved Scalability & Performance – Scalability with performance validated to 6,000 hosts, 48,000 cores and 200K pending jobs.		Yes
Ease of Administration Features		
Multiple Administrator Support	Yes	Yes
Platform LSF analytics support	Yes	Yes
Platform LSF license scheduler support	Yes	Yes
Application Encapsulation – application specific parameters can now be expressed in application profiles in order to simplify queue configurations.		Yes
Standardized treatment of memory/capacity units – units for all memory or capacity related parameters can be represented in KB, MB, GB or TB.		Yes
Dump daemon configurations at run-time – By dumping run-times parameters for LSF daemons at run-time it becomes easier for administrators to troubleshoot configuration related issues.		Yes
License scheduling improvements – including allocating license share by percentage rather than token counts and improved reporting of license status.		Yes
Superior Job Group Management – Allows for macro substitution when creating and managing job groups makes it easier to operate on groups of jobs at one time and save administration time and effort.		Yes
Improved configurability of user limits – a "default@" directive in lsb.users makes it easier to enforce granular resource related limits for groups and		Yes

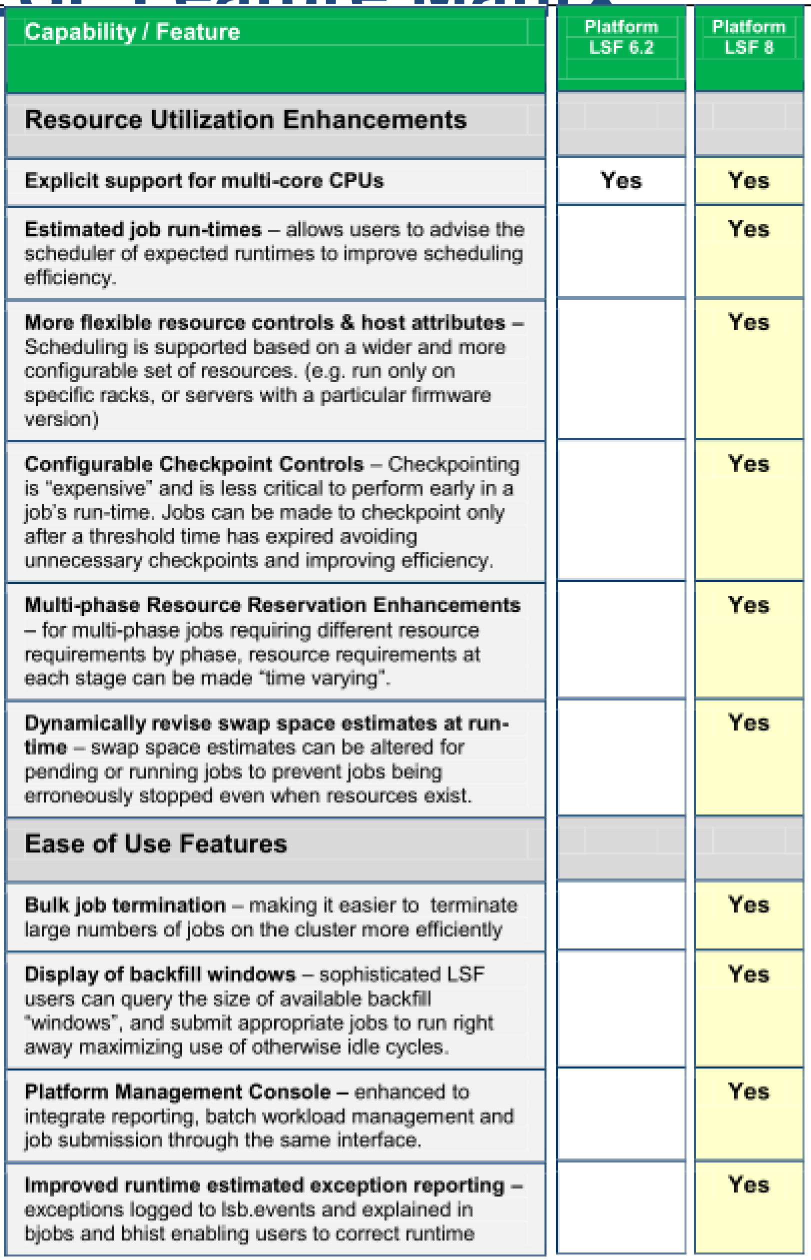
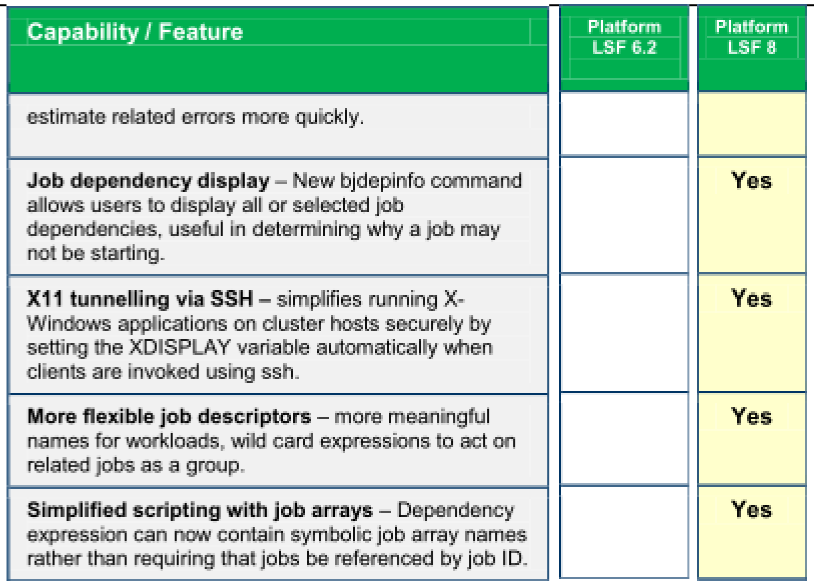
11

LSF Feature matrix

Capability / Feature	Platform LSF 6.2	Platform LSF 8
individual users.		
Auto-shutdown of LSF daemons in dynamic clusters – In clusters with dynamic hosts, where hosts need to be removed frequently from running clusters, LSF daemons on these hosts now shutdown automatically.		Yes
Delegation of Administrative Rights – LSF administrators can delegate job management tasks to selected group level administrators.		Yes
Automated OS version detection – An external static LIM script enables LSF to automatically detect OS types and versions, display them and incorporate them into resource requirement expressions.		Yes
Internal license usage display – makes it easier for LSF administrators to track LSF license features and usage.		Yes
Enhancements to Delegation of Administrative rights – delegated administrators can modify group membership and fairshare resource allocations as well as manage jobs.		Yes
Better parallel jobs resource reporting – Multiple improvements related to tracking resource utilization for parallel jobs by host and on aggregate enabling better reporting and smarter scheduling.		Yes
Cluster Availability, Reliability Features		
EGO management of LSF daemons for improved reliability – optional, depending on configuration.		Yes
Granular job re-queue policies – more configurability is provided when re-queuing failed jobs including the ability to take job exit code into consideration to make job execution more reliable.		Yes
Live Cluster Reconfiguration – new bconf facility enables change in cluster configurations "live" with no need to restart LSF daemons.		Yes

Capability / Feature	Platform LSF 6.2	Platform LSF 8
Improved Flexibility		
JSDL support – use XML job description semantics to express job parameters for compatibility between workload scheduler.		Yes
Support for multiple resource requirement strings – to simplify scripting and flexibility.		Yes
Host models increased from 128 to 1024 – new architecture detection algorithm ensures that the latest processors and hardware are correctly identified.		Yes
Job Groups with per-branch job slot limits – In hierarchical job group expressions, limits can be imposed at a "branch level" making it easier to express granular sharing policies between groups.		Yes
Various job submissions improvements – make jobs non-re-runnable even if the queue is, provide run-time estimates, working directories, job-level post-execution commands etc.		Yes
Mixed OS path support – makes it easier to enable jobs running across different operating systems by allowing path syntax to depend on the operating system.		Yes
Dynamically resizable jobs – multi-slot jobs such as MPI parallel jobs or session scheduler jobs can be re-sized on the fly enabling more efficient allocation for jobs that change geometry as they run (i.e. Avoid the "long tail" problem).		Yes
Blaunch support Parallel jobs on Windows – used enhanced blaunch mechanism to management parallel workloads on Windows 2000 and later.		Yes
More flexible data movement – Platform LSF native lsrcp mechanisms made scriptable so that users can plug in their own data transfer mechanisms (like scp to achieve secure copy between hosts).		Yes
Simplified password synchronization between Linux and Windows in mixed clusters – New lspasswd facility on Linux enables passwords to be changed on Windows cluster hosts.		Yes

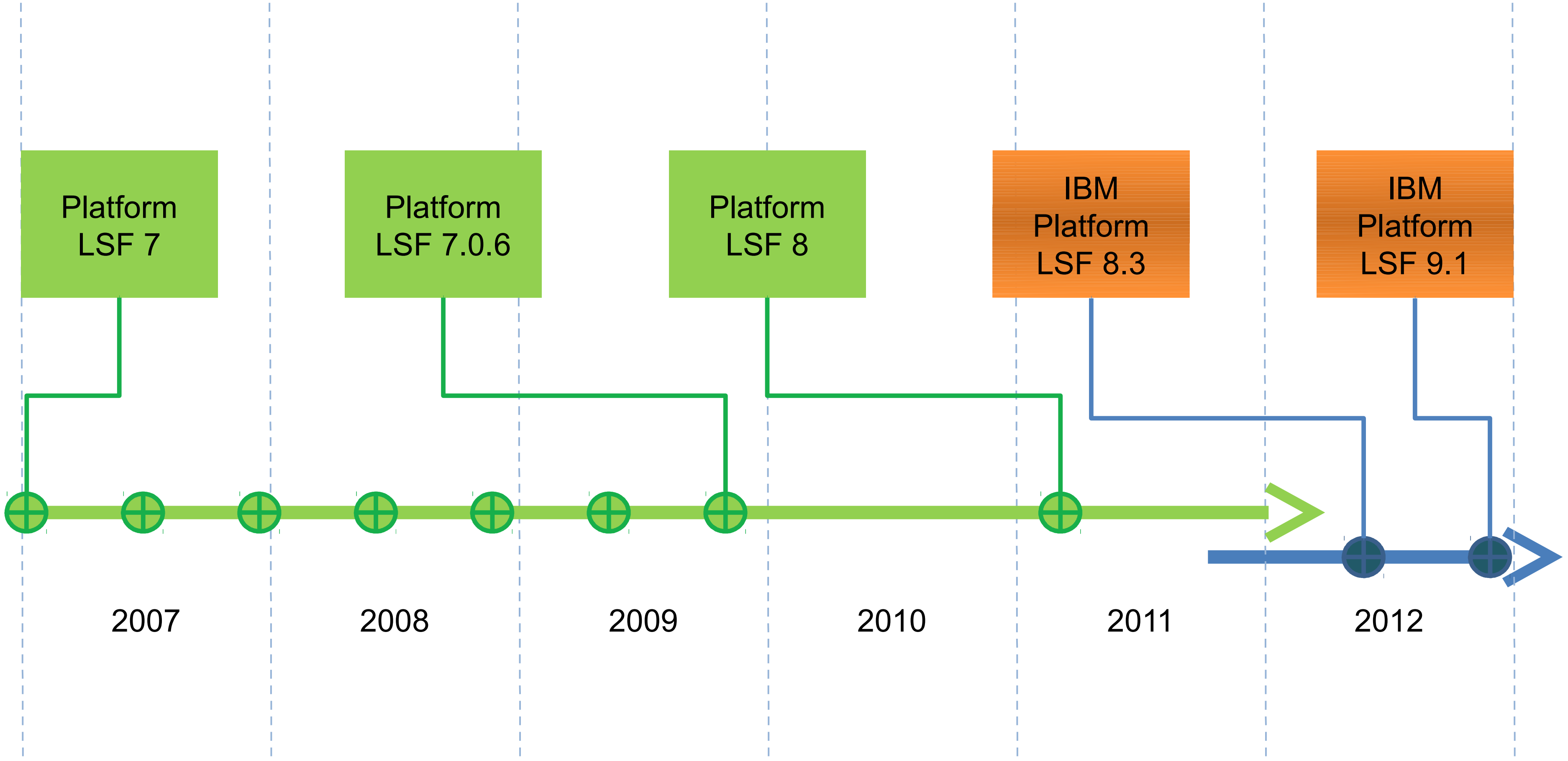
LSF Feature Matirx

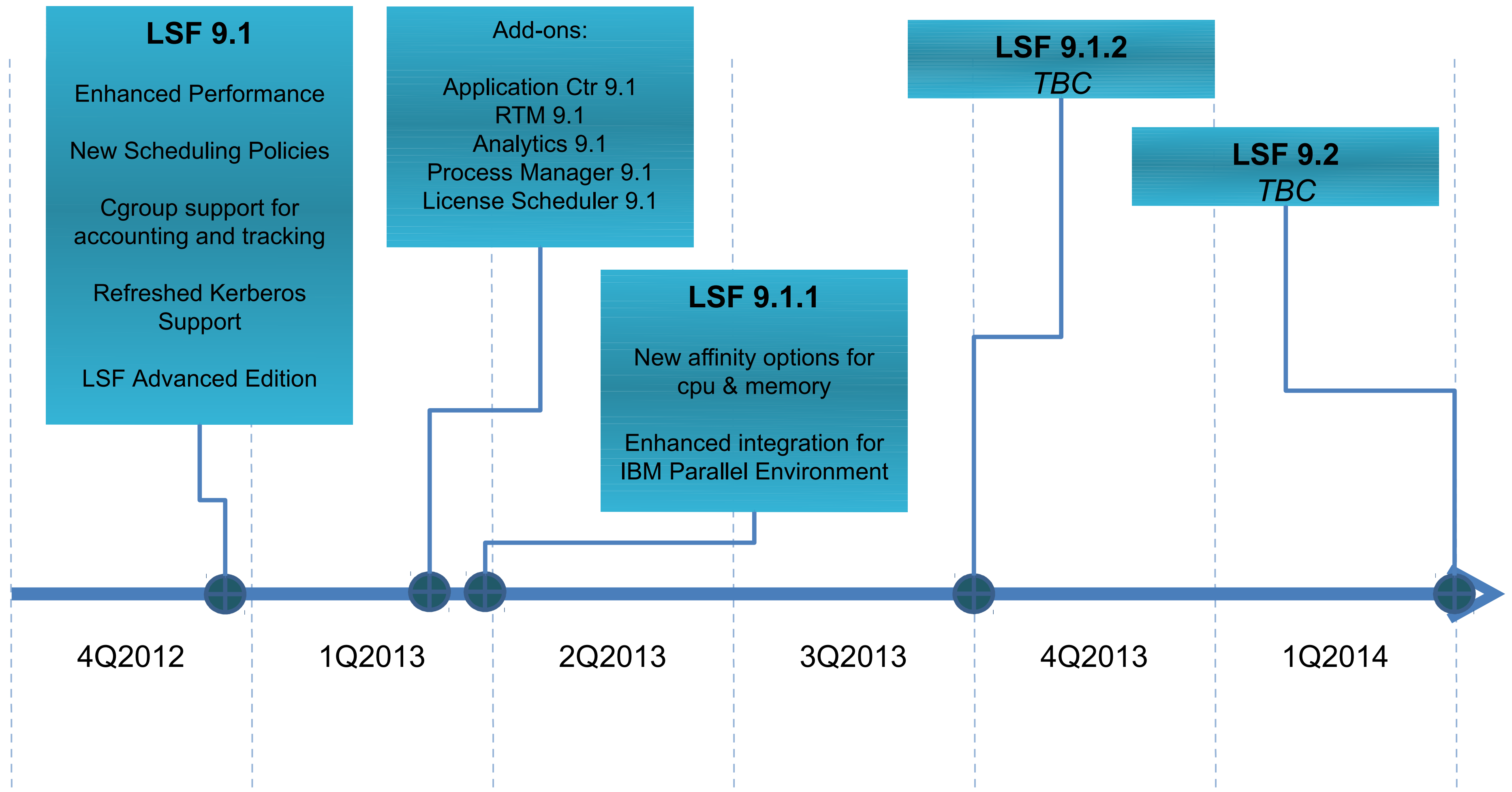
Capability / Feature	Platform LSF 6.2	Platform LSF 8
Resource Utilization Enhancements		
Explicit support for multi-core CPUs	Yes	Yes
Estimated job run-times – allows users to advise the scheduler of expected runtimes to improve scheduling efficiency.		Yes
More flexible resource controls & host attributes – Scheduling is supported based on a wider and more configurable set of resources. (e.g. run only on specific racks, or servers with a particular firmware version)		Yes
Configurable Checkpoint Controls – Checkpointing is "expensive" and is less critical to perform early in a job's run-time. Jobs can be made to checkpoint only after a threshold time has expired avoiding unnecessary checkpoints and improving efficiency.		Yes
Multi-phase Resource Reservation Enhancements – for multi-phase jobs requiring different resource requirements by phase, resource requirements at each stage can be made "time varying".		Yes
Dynamically revise swap space estimates at run-time – swap space estimates can be altered for pending or running jobs to prevent jobs being erroneously stopped even when resources exist.		Yes
Ease of Use Features		
Bulk job termination – making it easier to terminate large numbers of jobs on the cluster more efficiently		Yes
Display of backfill windows – sophisticated LSF users can query the size of available backfill "windows", and submit appropriate jobs to run right away maximizing use of otherwise idle cycles.		Yes
Platform Management Console – enhanced to integrate reporting, batch workload management and job submission through the same interface.		Yes
Improved runtime estimated exception reporting – exceptions logged to lsb.events and explained in bjobs and bhist enabling users to correct runtime estimate related errors more quickly.		Yes
Job dependency display – New bjdepinfo command allows users to display all or selected job dependencies, useful in determining why a job may not be starting.		Yes
X11 tunnelling via SSH – simplifies running X-Windows applications on cluster hosts securely by setting the XDISPLAY variable automatically when clients are invoked using ssh.		Yes
More flexible job descriptors – more meaningful names for workloads, wild card expressions to act on related jobs as a group.		Yes
Simplified scripting with job arrays – Dependency expression can now contain symbolic job array names rather than requiring that jobs be referenced by job ID.		Yes

13

High Level Timeline



High Level Roadmap



Key Features in Platform LSF 8

Goal: Simplify Administration

- **Simplified Administration through Dynamic Configuration**
- **Delegation of Administrative Rights**
- **Guarantee Resources via SLA’s**
- **Enhanced Fairshare Scheduling Policies**
- **Enhanced Pre-Emptive Scheduling Policies**
- **Misc Enhancements**
- **GPU Aware Scheduling**
- **…and performance and scalability**

Dynamic Configuration

- LSF7
  - LSF7 provides two methods of propagating configuration changes – reconfig and restart.
  - While reconfiguring is less obtrusive than restart, it can still take many minutes on large busy clusters.
  - This is quite an overhead when all you want to change is one parameter

- LSF8
  - In LSF8 we have provided a method to allow the most common configuration changes to be dynamically applied via a CLI/API without the need for reconfiguration or restart.

bconf

- bconf
- Submits live reconfiguration requests, updating configuration settings in active memory without restarting daemons.
- bconf is enabled when LSF_LIVE_CONFDIR is defined in lsf.conf.
- bconf action object_type=object_name "value_pair[;value_pair...]"] [-c "comment"] [-f]
  - *action* is the requested action supported by live reconfiguration. It can be one of: addmember, rmmember, update, create, add, delete.
  - *object_name* is the name of the existing object, or object being created.
  - *value_pair* is the key (object attribute) and allowed values used in a bconf request. It is of the form keyword=value, using the same keywords and syntax as in LSF configuration files. Not all LSF configuration keywords can be used with all actions.

Live reconfiguration: bconf

```
$bconf action object_type=object_name
  "key-value_pair[;key-value_pair...]"] [-c "comment"] [-f]
```

Object	Actions	Keys
User	update	MAX_JOBS, JL/P, MAX_PEND_JOBS
UserGroup	create, delete, update, addmember, rmmember	MAX_JOBS, JL/P, MAX_PEND_JOBS, GROUP_MEMBER, USER_SHARES, GROUP_ADMIN
Host	add, update	MXJ, JL/U, EXIT_RATE, io, it, ls, mem, pg, r15s, r1m, r15m, swp, tmp, ut, hostmodel, hosttype, resources
HostGroup	update, addmember, rmmember	GROUP_MEMBER
Queue	update, addmember, rmmember	UJOB_LIMIT, PJOB_LIMIT, QJOB_LIMIT, HJOB_LIMIT, FAIRSHARE
Limit	create, delete, update, addmember, rmmember	QUEUES, PER_QUEUE, USERS, PER_USER, HOSTS, PER_HOST, PROJECTS, PER_PROJECT, SLOTS, SLOTS_PER_PROCESSOR, MEM, TMP, SWP, JOBS, LICENSE, RESOURCE
GPool	update, addmember, rmmember	DISTRIBUTION

History and Auditing

- The original configuration files are not modified.
- Changes are held in separate files.
- An audit trail of who changed what is maintained.

```
$ bconf hist -l

Thu Sep 02 11:04:47 2010: Usergroup <ugroup2> deleted by user <hma>
               Changes made:
               Hostgroup <hgroup3> updated in <lsb.hosts> with <GROUP_ADMIN=[full]>
               Usergroup <ugroup3> updated in <lsb.users> with
<USER_SHARES=[default,10]>
               Usergroup <ugroup3> updated in <lsb.users> with <GROUP_MEMBER=ugroup1>
               Usergroup <ugroup3> updated in <lsb.users> with <GROUP_ADMIN=->
               Queue <normal> updated in <lsb.queues> with <ADMINISTRATORS=->
               Queue <normal> updated in <lsb.queues> with <USERS=ugroup1>
               Usergroup <ugroup2> deleted in <lsb.users>
```

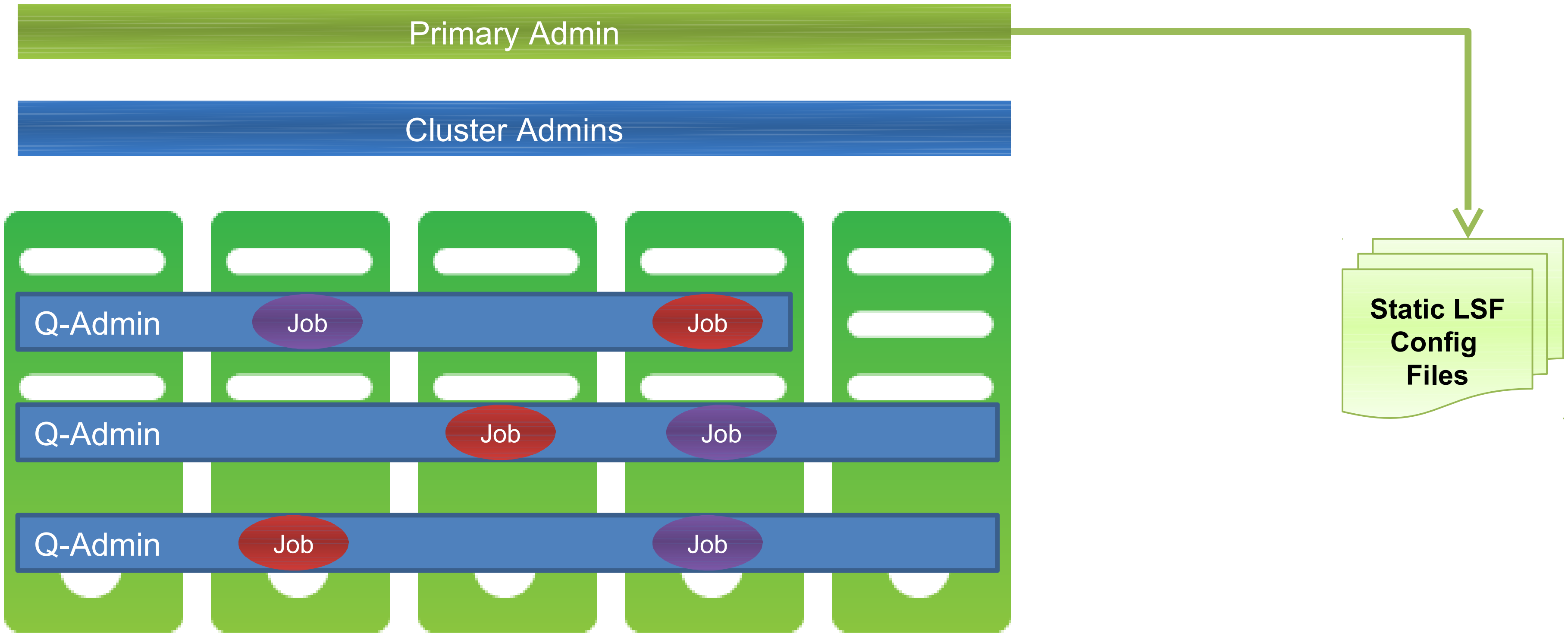
Delegation of Admin Rights

- In LSF 7:
  - We added the concept of a “user group admin”
  - This allowed Project Managers/Line of Business owners to control the workload of their project/BU.

- In LSF 8:
  - We have extended this concept to allow Project Managers/LoB owners to dynamically modify membership and to dynamically modify fairshare within the group.
  - This means the “LSF Admin” does not need to be concerned with “internal” project priority/membership changes.

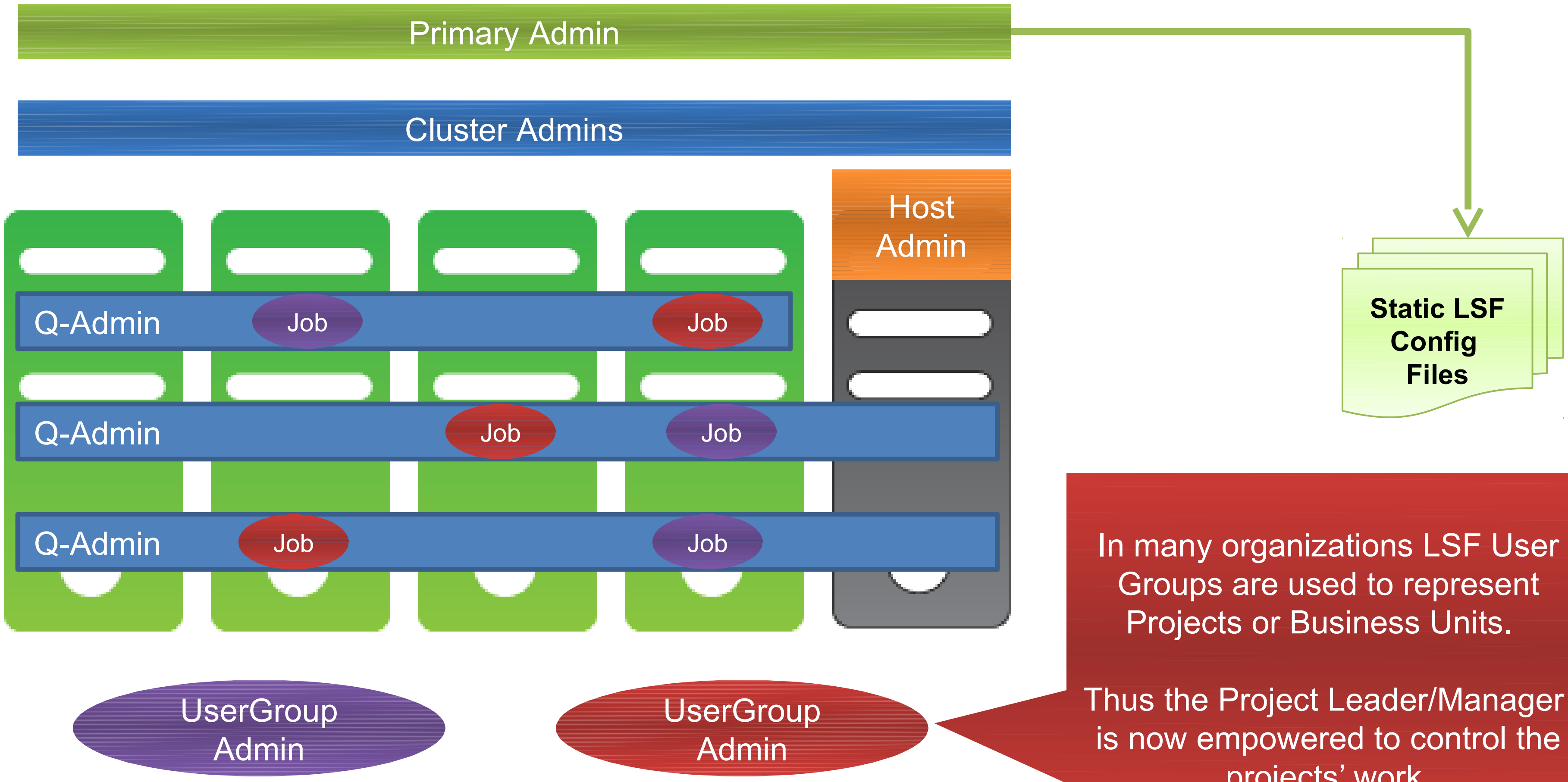
Administration Model in LSF6

Cluster and Queue Admins can control workload, but only the Primary Admin change the configuration.



Administration Model in LSF7

Cluster, Queue and UserGroup Admins can control workload, but not change the configuration.

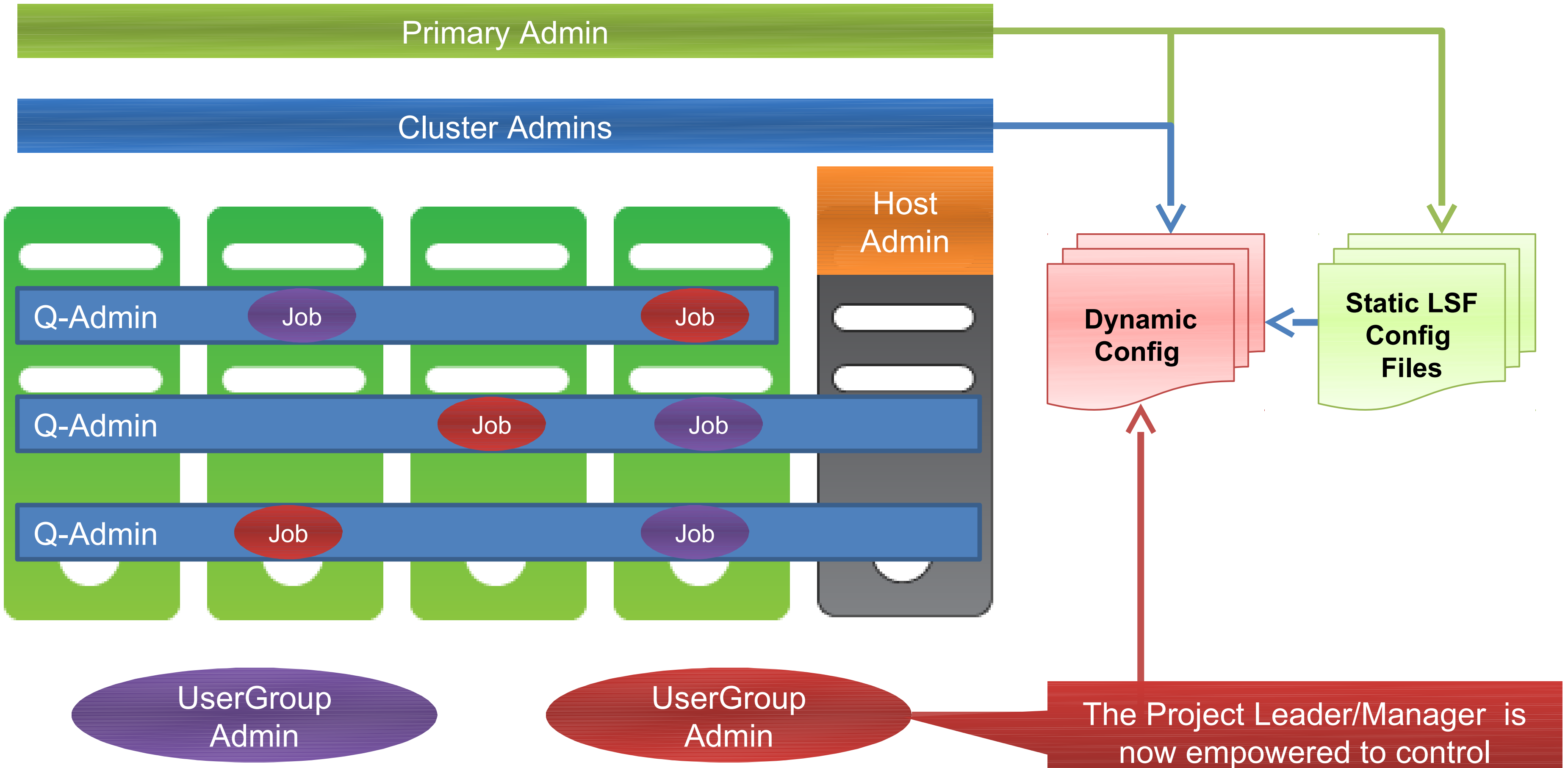


In many organizations LSF User Groups are used to represent Projects or Business Units.

Thus the Project Leader/Manager is now empowered to control the projects' work.

Administration Model in LSF8

Cluster Admins and optionally **UserGroup Admins** can now dynamically modify configuration.



The Project Leader/Manager is now empowered to control membership and fairshare within the project.

Administrative Scope

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER  USER_SHARES                 GROUP_ADMIN
DefGroup    (all)         ()                          ()
groupA      (user1 user2) ([user1,3] [user2,4])       (admin1)
groupB      (user3 user4) ([user2,1] [default,10])    (admin2[usershares])
groupC      (user3 user5) ([user2,1] [default,10])    (admin3[full])
End UserGroup
```

- In the example:
  - LSF7: Admin1 can control the jobs belonging to groupA
  - LSF8: Admin2 can control the jobs belonging to groupB, and adjust the shares within the group.
  - LSF8: Admin3 can control the jobs, adjust the shares, and alter the membership of the group.

Enforcing Membership

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER  USER_SHARES                  GROUP_ADMIN
DefGroup     (all)         ()                           ()
groupA       (user1 user2) ([user1,3] [user2,4])        (admin1)
groupB       (user3 user4) ([user2,1] [default,10])     (admin2[usershares])
groupC       (user3 user5) ([user2,1] [default,10])     (admin3[full])
End UserGroup
```

- DEFAULT_USER_GROUP=*group*
  - Jobs submitted without –G *group* will automatically be placed in the default group
- STRICT_UG_CONTROL=Y|N
  - In the above example, user3 is a member of both groupB and groupC. Which admin has the right to control those jobs?
  - With STRICT_UG_CONTROL enabled, the submission group (-G) will be used for access control.
  - So if the job was submitted as *–G groupB*, admin3 <u>would not</u> be able to control the job.

Guaranteed Resources

- This extends the existing SLA scheduling policy to guarantee resources to groups of jobs
    - Resources can be slots, whole hosts, or user-defined shared resources, e.g. licenses.
    - Jobs may be grouped by queue, user, fairshare group, license project and/or application.
    - Jobs can be automatically attached to a service class via access controls allowing Administrators to enable Guarantee’s without requiring end users to change submission procedures.

  – Provide access controls to who can use the SLA
  – Provide mechanisms to view the effectiveness of the policy

Guaranteed Resource

- Guaranteed resource pools are defined in lsb.resources and used by consumers defined within SLAs in lsb.serviceclasses.
  - Choose the type of resource pool to configure. The three possible types are hosts, slots, or resources (such as licenses).
  - Configure a GuaranteedResourcePool section in lsb.resources. Begin with the line Begin GuaranteedResourcePool and end with the line End GuaranteedResourcePool. Specify the following: NAME; the name of the guaranteed resource pool.
    - TYPE; the guarantee type (slots, hosts, or resources).
    - DISTRIBUTION; share assignments for all SLAs using the resource pool. Can be percent or absolute numbers.
    - Optional parameters for the GuaranteedResourcePool section are HOSTS, RES_SELECT, LOAN_POLICIES, DESCRIPTION, and SLOTS_PER_HOST.
  - Configure a ServiceClass section in lsb.serviceclasses for each SLA. Begin with the line Begin ServiceClass and end with the line End ServiceClass. For each SLA you must specify:
    - NAME; the name of the SLA.
    - GOALS = [GUARANTEE]
    - Optional parameters for the ServiceClass section are ACCESS_CONTROL, AUTO_ATTACH, and DESCRIPTION.

Use Case #1 – Service Level Objectives

- **Scenario**
  - Heterogeneous cluster, with hosts grouped by OS, and memory size.
  - Each business unit (BU) at site has its own queue.
  - Administrator must ensure a minimum number of compute resources for each BU, of each host class.
  - When there is demand from a BU, resources must be available within 10 minutes, to meet the SLA

- **Solution**
  - Configure one service class per BU: restrict access to jobs of the BU's queue.
  - Group hosts into host (or slot) pools, according to the relevant characteristics of OS and memory.
  - Configure guarantees in each pool, according to the service level objectives of the site. In each pool, set :

```
LOAN_POLICIES = QUEUES[all] DURATION[10]
```

allowing resource loans to jobs that are less than 10 minutes.

Use Case #2 – Exclusive Resources

- **Scenario**
  - Dedicated large memory hosts for large memory jobs (e.g. 128GB).
  - Currently, using exclusive resources - hosts can only be used by jobs that explicitly request them.
  - Often, the large memory hosts are idle.
  - Desire to share the large memory hosts with other jobs, when there is no demand from large memory jobs.

- **Solution**
  - Configure service class **SC** for large memory jobs.
  - Guarantee 100% of the big memory hosts to **SC**.
  - In the guaranteed host pool :

```
LOAN_POLICIES = QUEUES[all] SUSP_ON_DEMAND
```

  - Whenever there is demand for large memory jobs, LSF prevents any other jobs from starting on the hosts, allows large memory jobs to start ASAP.

Use Case #3 – Sharing LargeMem Hosts

- **Scenario**
  - The cluster contains a number of 16-core large memory hosts.
  - There is a large memory queue for large memory jobs.
    - Jobs in this queue use a single core (slot), and most of the memory the host.
  - Want to ensure that each BU can run some big memory jobs.
  - Want the remaining 15 cores on each big memory host used as well, to run small memory jobs (from other queues).

- **Solution**
  - Configure a service class for each BU, for jobs from the big memory queue.
  - Set guarantees for BUs, from the pool of big memory hosts.
  - Ensure that only 1 slot per host is reserved for guarantees:

```
SLOTS_PER_HOST = 1
```

  - Jobs from other queues are able to use the remaining 15 cores on each big memory host.

Fairshare Enhancements

- Fairshare is a very powerful and commonly used policy.
- While different shares can be configured, the parameters affecting the formula are set globally in lsb.params:
  - RUN_TIME_FACTOR
  - CPU_TIME_FACTOR
  - ENABLE_HIST_RUN_TIME
  - HIST_HOURS
  - FAIRSHARE_ADJUSTMENT_FACTOR
  - RUN_JOB_FACTOR
  - COMMITTED_RUN_TIME_FACTOR
  - RUN_TIME_DECAY
- In LSF 8 these parameters can also be defined at the queue level allowing even greater flexibility.

Fairshare Scheduling

- In case of resource contention, it divides access to cluster resources among users and groups by assigning shares
- Used to prevent any one user from monopolizing cluster resources.
- Fairshare scheduling types:
  - Queue (Equal Share, User Hierarchical)
  - Cross-Queue
  - Host Partition

Fairshare Scheduling (cont.d)

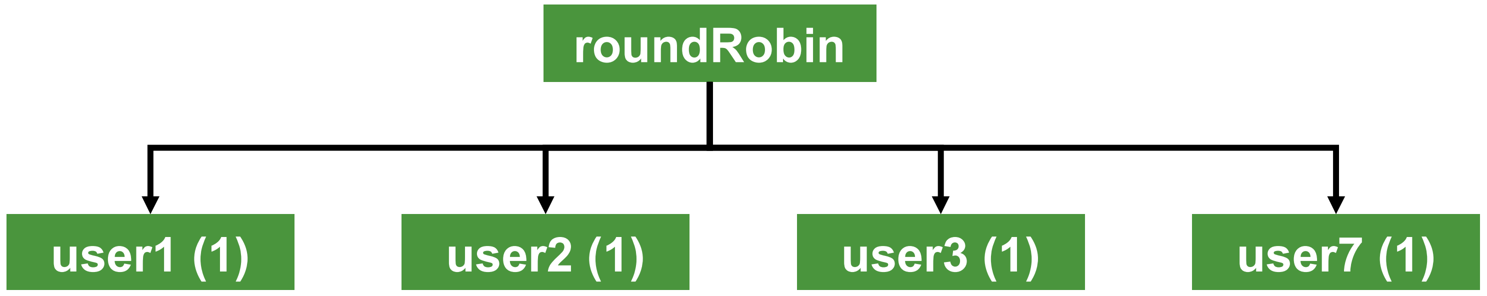
- Fairshare calculates a dynamic scheduling priority for each user
- The dynamic priority is calculated using:
  - Defined user share assignments
  - Number of job slots reserved and in use
  - Run time of running jobs
  - Cumulative actual CPU time
  - Historical run time of finished jobs (over **`HIST_HOURS`**)

Fairshare: Equal Share

- Balances resource usage equally between users
- Also called round-robin scheduling
- Defined in **lsb.queues**:

```
Begin Queue
QUEUE_NAME = roundRobin
PRIORITY = 40
FAIRSHARE = USER_SHARES[[default,1]]
USERS = userGroupA
End Queue
```

Fairshare: Equal Share (cont.d)



Fairshare: User Hierarchical

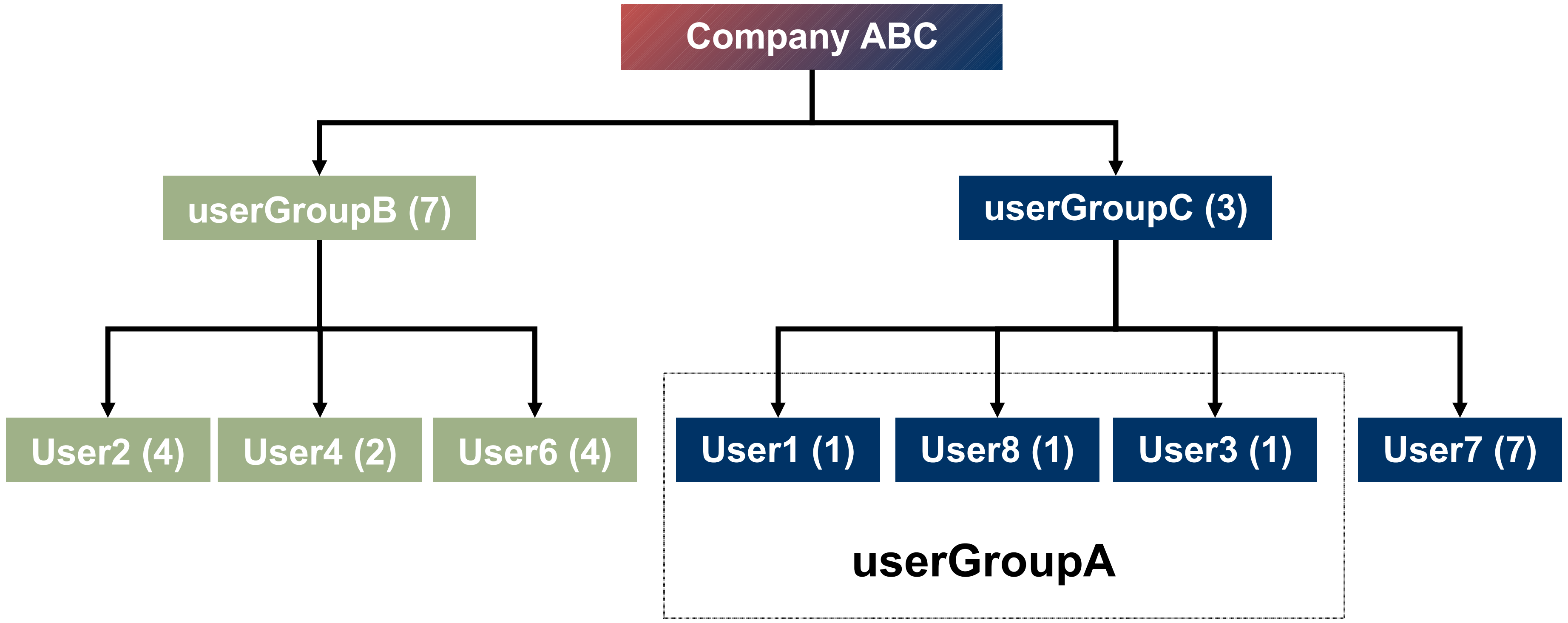
- Defined in the **lsb.users**:

```
Begin UserGroup
GROUP_NAME          GROUP_MEMBER          USER_SHARES
userGroupA    (user1 user2 user3)              ()
userGroupB    (user2 user4 user6)         ([user4,2] [default,4])
userGroupC    (userGroupA user7)          ([userGroupA@,1] [user7,7])
End UserGroup
```

- Defined in the **lsb.queues**:

```
Begin Queue
QUEUE_NAME = hierarchicalShare
PRIORITY = 40
USERS = userGroupB userGroupC
FAIRSHARE = USER_SHARES[[userGroupB,7] [userGroupC,3]]
End Queue
```

Fairshare: User Hierarchical (cont.d)



Fairshare: Host Partition

- A user’s priority is calculated cluster wide
- Handles resource contention across *multiple queues*
- Defined in **lsb.hosts**:

```
Begin HostPartition
HPART_NAME = Partition1
HOSTS = training1 training2
FAIRSHARE = USER_SHARES[[userGroupA@,5] [userGroupB,10]]
End HostPartition
```

Fairshare: Cross-Queue

- Handles resource contention across multiple queues
- Applies the same fairshare policy across several queues for a user
- The fairshare policy is defined in the "master" queue
- The "slave" queues are identified in the "master" queue and inherit the fairshare policy from the "master“
- Several sets of master-slave queues can be defined
- A queue cannot belong to multiple master-slave sets

Fairshare: Cross-Queue (cont.d)

```
Begin Queue
QUEUE_NAME = verilog
DESCRIPTION = master queue definition cross-queue
PRIORITY = 50
FAIRSHARE = USER_SHARES[[user1,100] [default,1]]
FAIRSHARE_QUEUES = normal short
HOSTS = hostGroupC # resource contention
RES_REQ = rusage[verilog = 1]
End Queue
```

```
Begin Queue
QUEUE_NAME = short
DESCRIPTION = short jobs
PRIORITY = 70 # highest
HOSTS = hostGroupC
RUNLIMIT = 5 10
End Queue
```

```
Begin Queue
QUEUE_NAME = normal
DESCRIPTION = default queue
PRIORITY = 40 # lowest
HOSTS = hostGroupC
End Queue
```

Cross-Queue Fairshare Dispatching

- By default, the user's fairshare priority defines job dispatching order and not queue priority
- To force job dispatching in order of queue priority, define ***`DISPATCH_ORDER`***

Cross-Queue Fairshare Dispatching (cont.d)

```
Begin Queue
QUEUE_NAME = verilog
DESCRIPTION = master queue definition cross-queue
PRIORITY = 50
FAIRSHARE = USER_SHARES[[user1,100] [default,1]]
FAIRSHARE_QUEUES = normal short
DISPATCH_ORDER = QUEUE
HOSTS = hostGroupC # resource contention
RES_REQ = rusage[verilog = 1]
End Queue
```

```
Begin Queue
QUEUE_NAME = short
DESCRIPTION = short jobs
PRIORITY = 70 # highest
HOSTS = hostGroupC
RUNLIMIT = 5 10
End Queue
```

```
Begin Queue
QUEUE_NAME = normal
DESCRIPTION = default queue
PRIORITY = 40 # lowest
HOSTS = hostGroupC
End Queue
```

Queue-Based Fairshare

- **Queue-based fairshare** allows flexible slot allocation per queue as an alternative to absolute queue priorities by enforcing a soft job slot limit on a queue

- Allows users and admins to organize the priorities of work and tune the number of jobs dispatched from a queue.

- This ensures no single queue monopolizes cluster resources leaving other queues waiting to dispatch jobs

Queue-Based Fairshare

- How it Works:
- Allows flexible slot allocation per queue as an alternative to absolute queue priorities
- Queues belong to a pool, which is a named group of queues using the same set of hosts
- The total number of job slots in a pool is constant
- Queues participating in a queue-based fairshare pool cannot be preemptive or preemptable
- Refer to the LSF 8 documentation for details

Queue-Based Fairshare (cont.d)

```
Begin Queue
QUEUE_NAME = verilog
DESCRIPTION = verilog queue
PRIORITY = 50
SLOT_POOL = poolA
SLOT_SHARES = 50
HOSTS = hostGroupC # resource contention
RES_REQ = rusage[verilog = 1]
End Queue
```

```
Begin Queue
QUEUE_NAME = short
DESCRIPTION = short jobs
PRIORITY = 70 # highest
SLOT_POOL = poolA
SLOT_SHARES = 30
HOSTS = hostGroupC
End Queue
```

```
Begin Queue
QUEUE_NAME = normal
DESCRIPTION = default queue
PRIORITY = 40 # lowest
SLOT_POOL = poolA
SLOT_SHARES = 20
HOSTS = hostGroupC
End Queue
```

Enhanced control of Pre-Emption

In busy clusters, lower priority jobs may get pre-empted again and again, resulting in very long turnaround times.

- In LSF7 we introduced three parameters to help mitigate this:
  - MAX_JOB_PREEMPT=n
    - The maximum number of times a job can be pre-empted
  - NO_PREEMPT_FINISH_TIME=t
    - Do not pre-empt the job if it is near it's finish time
  - NO_PREEMPT_RUN_TIME=t
    - Do not consider this job for preemption once it has accumulated the specified run time

- In LSF 8 we have introduced two additional parameters to provide further control
  - NO_PREEMPT_INTERVAL=t
    - How long a job can run before it can be preempted again.
  - MAX_TOTAL_TIME_PREEMPT=t
    - Maximum accumulated time the job can be preempted. Once exceeded, the job will not be preempted again.
- These parameters are supported in lsb.params, lsb.queues and lsb.applications

Misc Enhancements

- $lspasswd
  - Now allows you to update your Windows password from Linux
- $bstop –r –s TSTP
  - New –r option to release slots on suspension
  - New –s option to send SIGTSTP rather than SIGSTOP
- ADVRSV_USER_LIMIT=X
  - Configurable number of max advance reservations
- LSB_SUBK_SHOW_EXEC_HOST=Y|N
  - Control whether the hostname is printed on stdout for bsub -K
- PREEXEC_EXCLUDE_EXIT_VALUES
  - Behaves the same as job exclude exit values, but applies to the pre-exec command.
- LSB_DISPLAY_YEAR
  - Toggles display of the year in the time string displayed by the commands bjobs –l, bacct -l, and bhist -l|-b|-t

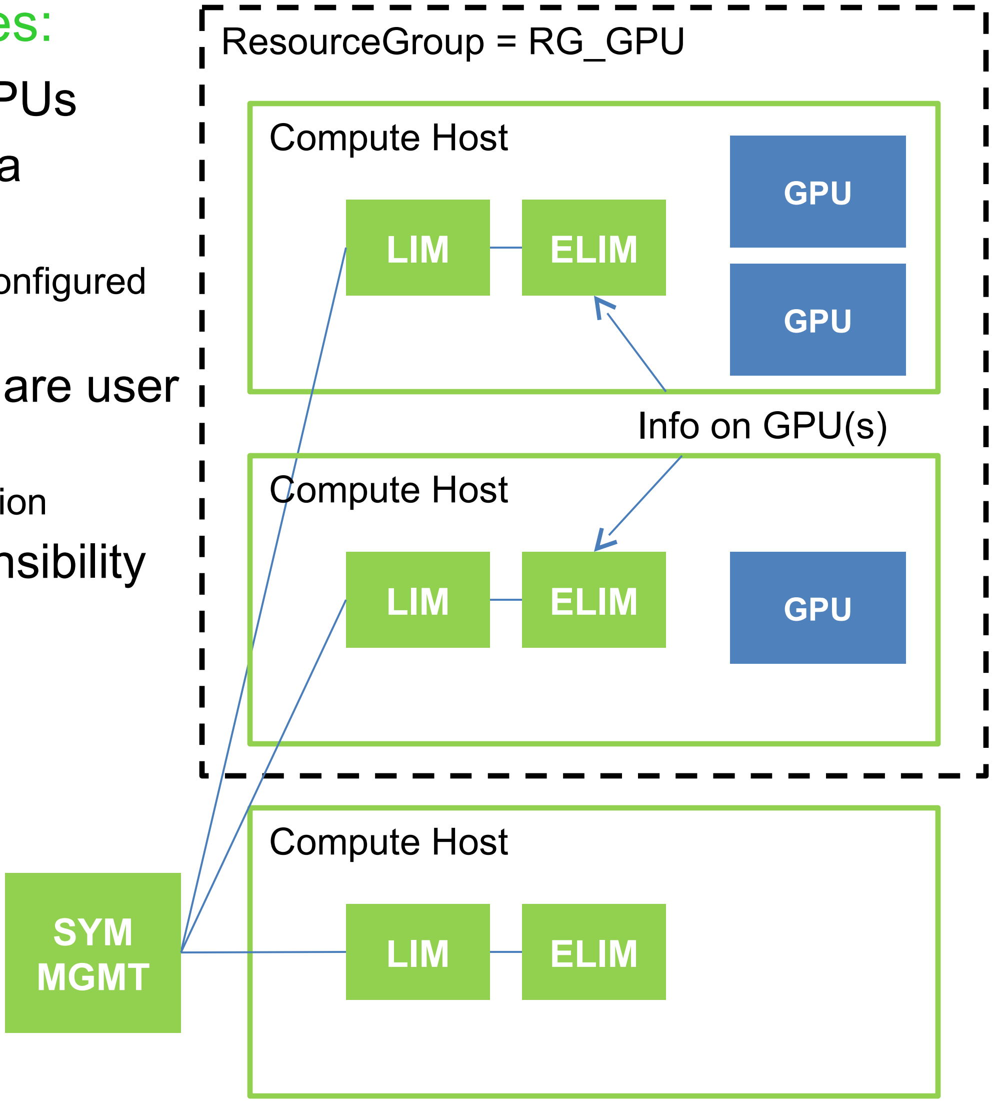
IBM Platform LSF GPU aware scheduling

Platform LSF GPU elim provides:

- Monitoring and detection of GPUs
- Group hosts with GPU(s) into a resource group
  - GPU-enabled applications can be configured to use this resource group
- Compute slots on these hosts are user configured
  - No fine grain GPU level core allocation
- GPU-enablement is the responsibility of the application developer

GPU Management:
- # of GPU
- # of GPU in “normal”
- # of GPU in “exclusive”
- # of GPU in “prohibited”

GPU Monitoring:
- Mode (normal, exclusive, prohibited)
  - Temperature
  - ECC error count



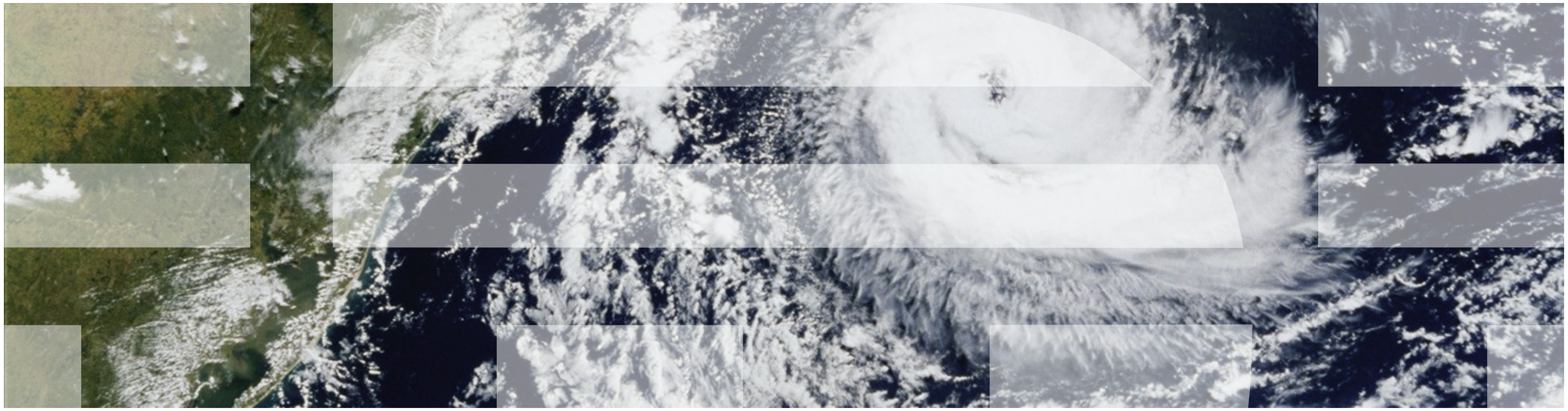
Scalability

- LSF 6.2
  - 3K hosts, 100K pending jobs
- LSF 7
  - 6K hosts, 24K slots, 100K pending jobs
- LSF 8
  - 6K hosts, 48K slots, 200K pending jobs
- LSF 8 100K Solution
  - 100K slots, 1.5M pending jobs

These are based on an EDA style, high throughput workload – 48,000 running jobs, each a few minutes in duration.
For parallel/HPC workloads with longer job duration, much higher scalability is achievable.

IBM Platform LSF 9.1

Performance Enhancements



IBM Platform LSF 9.1 Key Enhancements

- Increased Performance and Scalability
  - Query Response Improved, Scheduling cycle decreased, memory optimization, decrease start/restart time, parallel startup/restart
- Usability & Manageability
  - Clearer reporting of resource usage
  - Fast detection of hung hosts/jobs, working directory management
  - Leverage kernel cgroup functionality to replace/improve existing functionality for Process Tracking
  - New CPU and Memory Affinity functionality (9.1.1)
- Scheduling
  - Alternative/time based resource requirements
  - SLA scheduling extended to consider memory in addition to cores
  - Numerous MultiCluster enhancements (configuration, display of forwarded jobs, visibility of remoteID's, job priority, cluster preference, weighting, numeric/string resources).
- Parallel Job Support
  - Re-architected integration with IBM PE (9.1.1)

Performance Enhancements

LSF scheduling performance enhancements

- Job dispatch by queue
  - To speed up interactive job dispatching, the scheduler now can broadcast scheduling decisions for each queue instead of waiting for the entire scheduling cycle complete.
- Optimized fair-share algorithm
  - For a very large fair-share tree (e.g. 4K user group, 500 users with -g), job election performance has been improved 10x.

Scheduler memory optimization

- Memory footprint reduced by 30~40%

Optimize job dependency evaluation

- In order to speed up each scheduling cycle, the number of job dependencies evaluated within each scheduling cycle is now configurable.
- This limits the time spent re-evaluating job dependencies when there is a large number of pending jobs in the system.

Faster Restarts

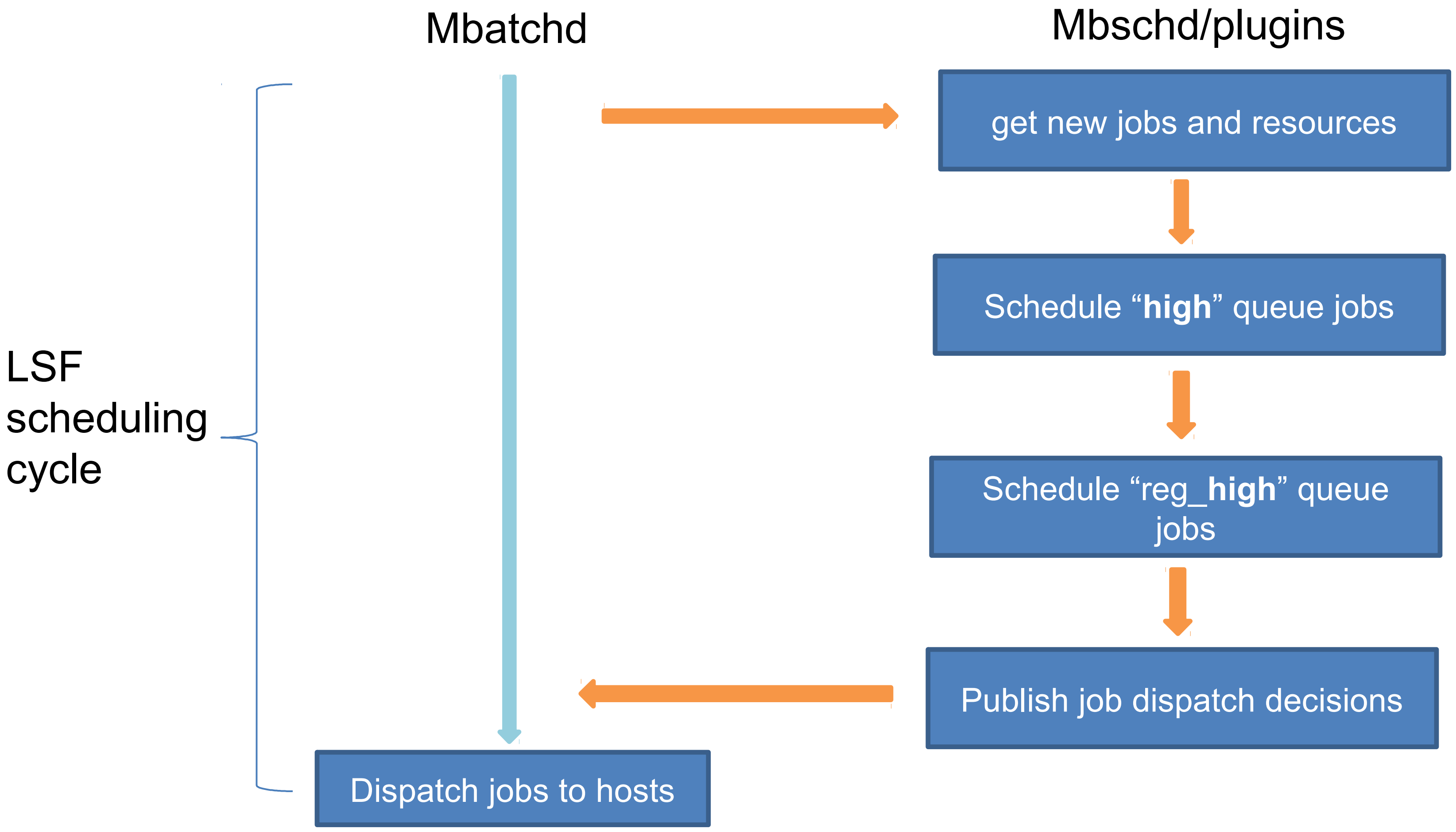
- New scheduler is brought up before the old one is killed

Query Performance

- Existing multi-thread queries mechanism has been enhanced to support many common LSF queries. Overall, with new enhancements, in a large busy cluster, LSF reduces query command response time from 1~2 seconds to 10~20 ms.
- Much of the content filtering is now moved to the server side
- Diagnostic functions to identify user/hosts that are generating significant network traffic

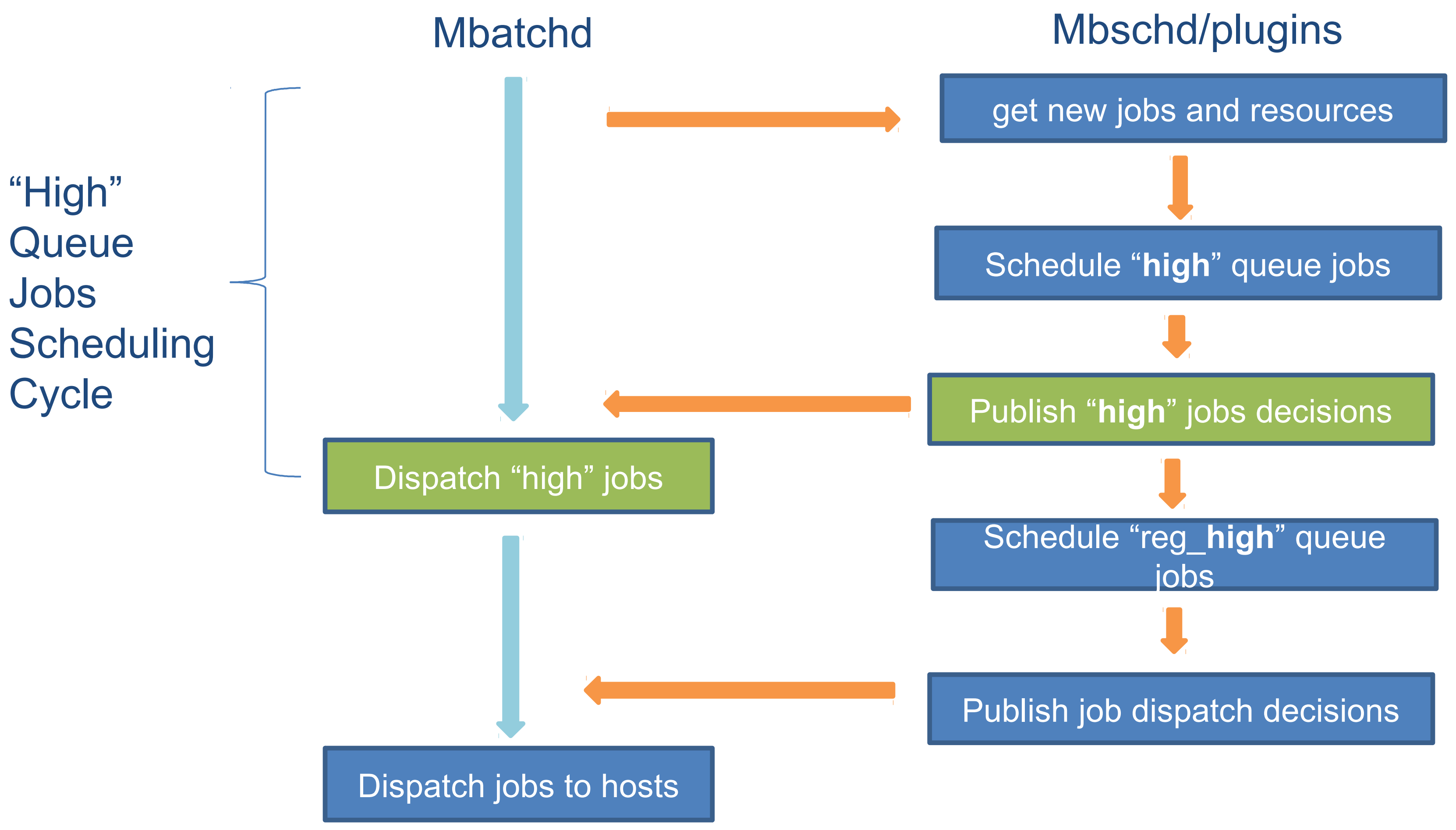
Shorten Scheduling Cycle for dispatching

- Current LSF scheduler publishes job scheduling decisions at the end of each scheduling cycle.

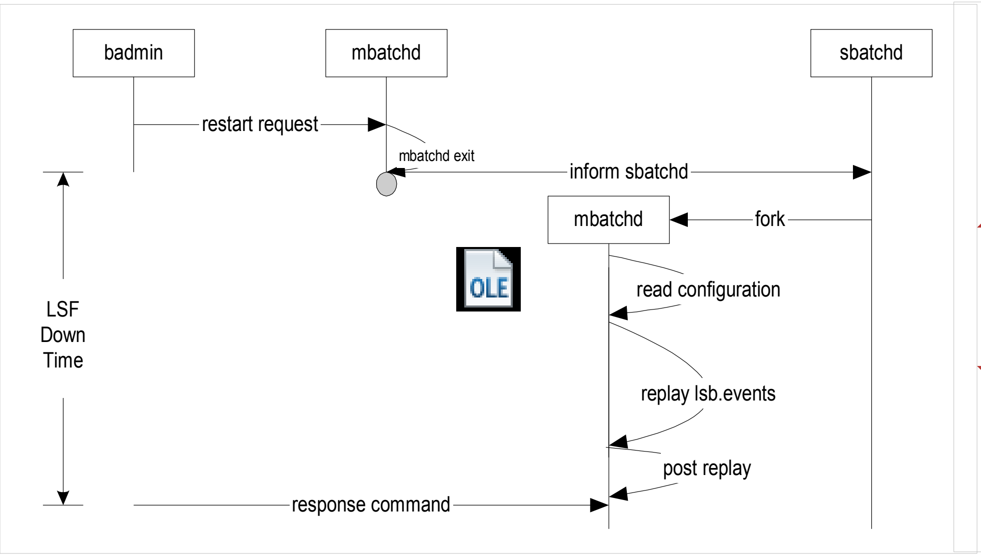

Mbatchd
Mbschd/plugins
get new jobs and resources
Schedule "**high**" queue jobs
Schedule "reg_**high**" queue jobs
Publish job dispatch decisions
Dispatch jobs to hosts
LSF scheduling cycle

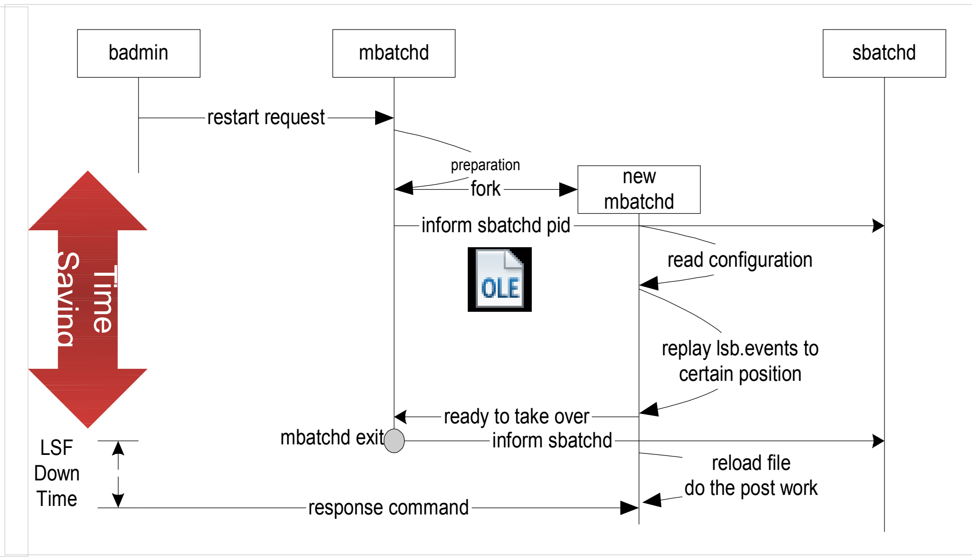
Shorten Scheduling Cycle for dispatching

- Publish job dispatch decisions in the middle of the scheduling cycle on a per queue basis. Enabled with a queue parameter DISPATCH_BY_QUEUE=Y.


Mbatchd
Mbschd/plugins
"High" Queue Jobs Scheduling Cycle
get new jobs and resources
Schedule "high" queue jobs
Publish "high" jobs decisions
Dispatch "high" jobs
Schedule "reg_high" queue jobs
Publish job dispatch decisions
Dispatch jobs to hosts

Parallel Restarts





Reducing bjobs query network traffic

- In very busy clusters, the volume of generated query data can potentially saturate the network connection.
- Prior to 9.1, the majority of filtering/formatting was performed on the client side – a lot of unnecessary data was being sent over the network.
- In LSF 9.1, we have enhanced the API such that the filtering is performed on the server side, and now only returns the information that is required for display.
- We have also added a new “-sum” to bjobs

```
zhding@hpdl05-103: bjobs –P newtest –u cchen  -sum
RUN     SSUSP   USUSP    UNKNOWN     PEND       FWD_PEND
100     0       0         0           700        0
```

Diagnosing Query Requests

- We have added a diagnostics routine to assist the Admin in identifying users/hosts that are generating high query loads.

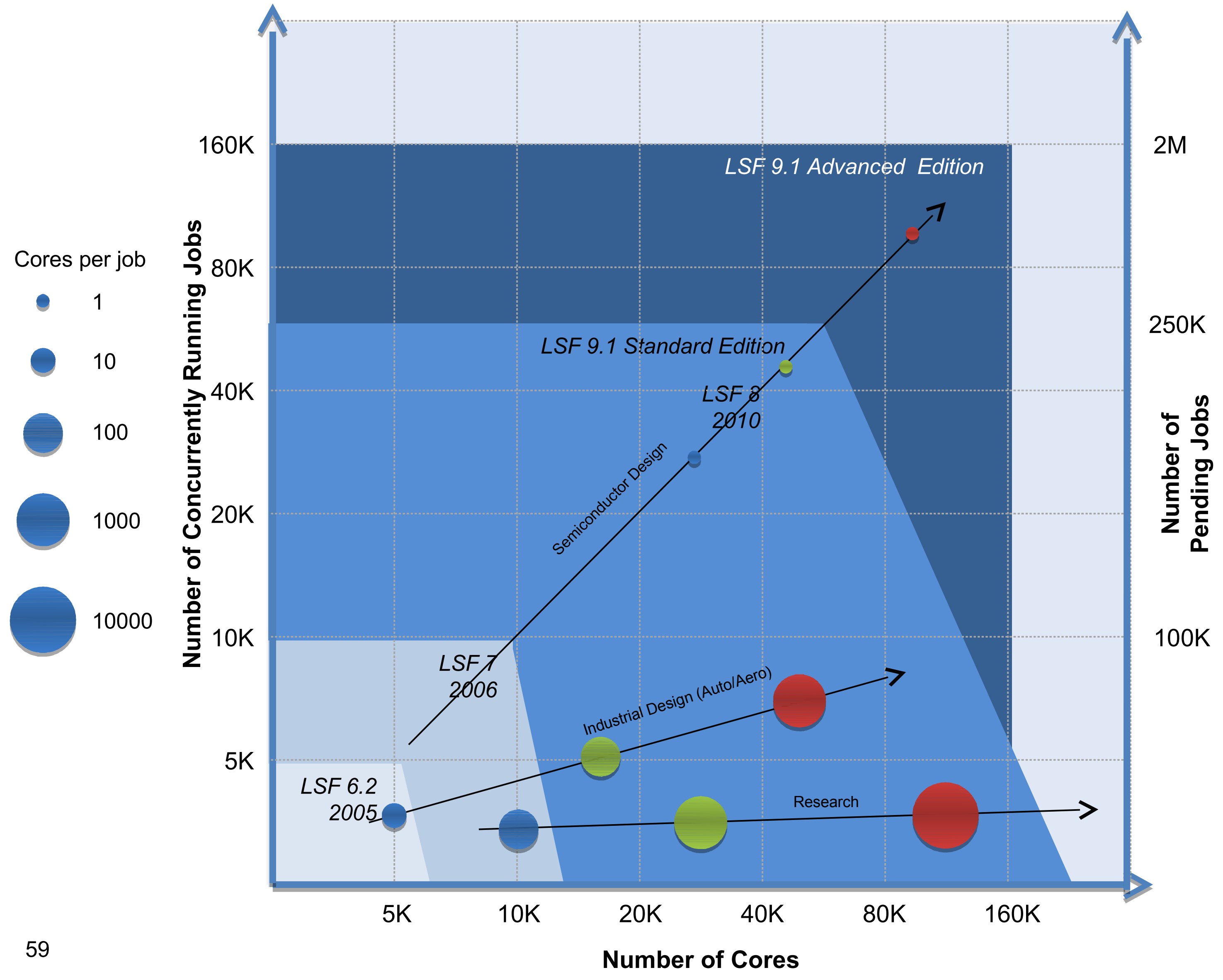
```
$ badmin diagnose –c query [[–f  logfile_name] [–d duration] |  [-o]]
```

- The output of log_file will be as below:

```
14:13:02 2011,bjobs,delpe02,sbadugu,1020, 0x0001
```

1. The time mbatchd processed the request
2. Command that generated the query
3. The source – hostname or IP
4. The user (if available)
5. Size of the query in KB
6. Query options bitmap - see lsbatchd.h file to get detail bits.

IBM Platform LSF Scalability (v9.1)

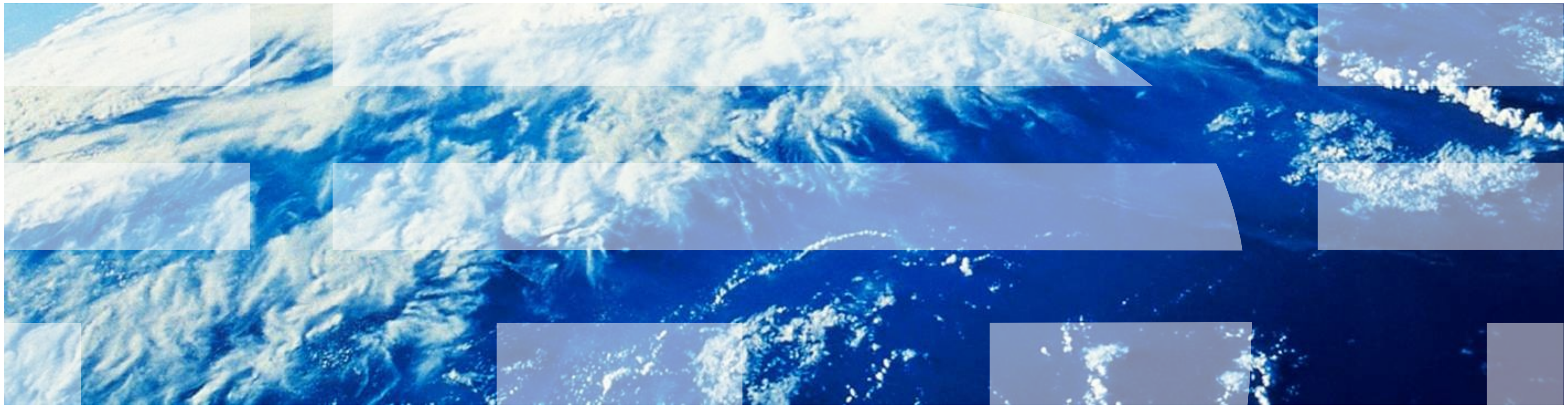


EDA Workloads:

- **LSF - Standard Edition**
  - 6K Nodes
  - 60K Cores/Concurrent Jobs
  - 250K pending jobs
  - Support a submission rate > 200 jobs/second
  - Support an average query time of less than 1 second.
- **LSF – Advanced Edition**
  - 18K Nodes
  - 160K Cores / Concurrent Jobs
  - 2M Pending Jobs
  - Support a submission rate > 400 jobs/second
  - Support an average query time of less than 1 second.

IBM Platform LSF 9.1

Usability & Manageability



Usability & Manageability

- Updated Hardware/topology detection
  - Use of hwloc library
- Cluster Status summary
- Display of real job RES_REQ
- Enhanced memory usage display
  - Peak and average, delta between asked for and used.
- Cgroup enhancement for process tracking and memory/cpu accounting

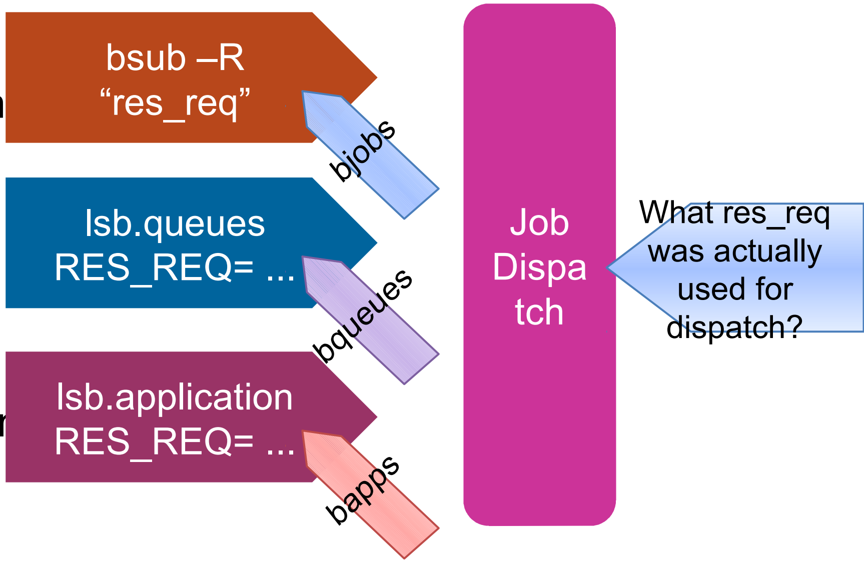
- Built-in resources slots/maxslots
- “bjobs –UF” for unformatted output
- Enhance cluster start up, detecting unavailable host and unknown job
  - 3x faster detection on startup
  -
- Hung job management
- Working Directory Management
- Updated Kerberos Support

Cluster Status Summary

```
$ badmin showstatus
LSF runtime mbatchd information
    Available local hosts (current/peak):
        Clients:                        2/2
        Servers:                        189/189
           CPUs:                        384/384
           Cores:                       1940/1940
           Slots:                       1920/1920

    Number of servers:                  191
        Ok:                             93
        Closed:                         96
        Unreachable:                    0
        Unavailable:                    2

    Number of jobs:                     728
        Running:                        544
        Suspended:                      0
        Pending:                        170
        Finished:                       14

    Number of users:                    77
    Number of user groups:              0
    Number of active users:             75

    Latest mbatchd start:               Sun Nov 25 14:22:01 2012
    Active mbatchd PID:                 3411125
```

Usability: Displaying the real RES_REQ

- Usage:
  - BJOBS_RES_REQ_DISPLAY=none | brief | full
  - none: bjobs -l does not display any level of resource requirement.
  - brief: bjobs -l only displays the combined an effective resource requirements. :
    - REQUESTED RESOURCES:
    - Combined : res_req
    - Effective : res_req
  - full: bjobs -l displays the job, app, queue, combined and effective resource requiremer
    - REQUESTED RESOURCES:
    - Job-level : res_req
    - App-level : res_req
    - Queue-level: res_req
    - Combined : res_req
    - Effective : res_req



  - bjobs will now show peak and average

Enhanced Process Tracking

LSF PIM

PIM daemon:

- Periodically scans through /proc file system
- Job runtime resource usage are collected from /proc/<pid>/{stat,smaps}.

Challenges:

- Run away process
  - Sometime processes escape via creative use of ssid etc
  - Thus termination/suspend/resume may miss some processes
- Short running process
  - Processes shorter than the sample interval are missed
  - Job memory usage is not accurate.
- Time consuming for systems with 1000's of processes

Kernel Control Group

Cgroups:

- Introduced after 2.6.24 (e.g. RHEL6, SuSE 11)
- Cgroups provides multiple subsystems for resource limits, accounting, isolation and control etc.
- Utilize cgroups "freezer", "cpuacct" and "memory" subsystems
  - Create cgroup per job and attach job processes into cgroup
  - Use cgroup to track job processes and helps on job control and resource usage collection.

Benefits:

- Support more stable and reliable job control
- Support more accurate light weight cpu and memory accounting even for run away and short job processes.
- Note: cgroup does incur some overheads
- NOTEs: Cadence fixing Virtuoso (registered trademark) platform to work under cgroup.

Flexible Current Working Directory (CWD)

- New options to automatically create a working directory for the job based on a pre-defined pattern

  - **bsub –cwd /filer12/scratch/%G/%U/%J**
  - **LSB_JOB_CWD** – environment variable,
  - Application profile based **JOB_CWD** parameter
  - Cluster wide CWD (**DEFAULT_JOB_CWD**),

All CWD parameters can includes the following dynamic patterns :

- %J - job id
- %JG - job group, if not specified job group it will be ignored
- %I - index, the default value is 0
- %EJ - execution job id
- %EI - execution index
- %P - project name
- %U - user name
- %G – users group

– TTL can optionally be specified in the application profile, or globally

– Also new –outdir option which takes the same format, TTL is NOT

IBM Platform LSF 9.1
Enhanced Scheduling Policies

Scheduling Enhancements

- Alternative Resource Requirements
  - Sometimes a user wants to run a job with one set of resources, or another
  - Sometimes a user wants a certain set of resources, but if they can't get those resources in timely manner, they are willing to consider alternative resources.

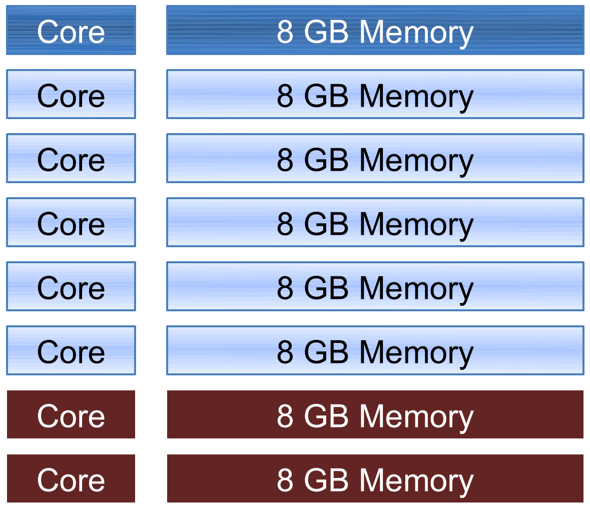
  - Alternative Resource Requirements provides this functionality:
  - $bsub-R "{ select[newmachines] } || { select[oldmachines] }@10"

- Memory Preemption

  - PREEMPTABLE_RESOURCES = mem

GSLA Scheduling: More Complex Guarantees

- In 8 we supported guarantees for slots (cores)
- In 9.1 we are extending this to support multi-dimensional "packages"
  - A "package" is a combination of slots & memory
  - A job requiring 1 core and <=8GB will use 1 "package"
  - A single core job requiring 16GB would use 2 packages



MultiCluster Enhancements

- Bjobs –fwd to display forwarded jobs

  Bjobs has been enhanced to introduce a new –fwd option to display forwarded jobs from submission cluster.

  - Visibility of remote job ID

  Bjobs, bhist and bacct have been enhanced to display both submission and execution job ids.

  Bjobs has been enhanced to support querying job information based on submission job id and submission cluster name on execution cluster

  - Recall a forwarded job
    - Use brequeue –p to recall a forwarded job from remote execution cluster and re-schedule from submission cluster.

  - Multi-cluster Job Priority Support
    - Send initial job priority (-sp) to remote execution cluster when job is forwarded

MultiCluster Enhancements

- Remote job modification
  - Enhance LSF to support modifying a forwarded job from submission cluster.
- Asked Cluster
  - Enhance LSF MC to support bsub –clusters option to allow user to define candidate clusters for job forwarding scheduling.
- Cluster Preference
  - Support cluster preference in both bsub –clusters and SNDJOBS_TO parameter in lsb.queue. This allows user to define which cluster LSF should consider first during forwarding.
- Slot Threshold on Receive Queue
  - Introduce IMPT_SLOTBKLG parameter in lsb.queues of execution cluster to define slot threshold of receive queue. Jobs from submission cluster will not be forwarded once threshold is reached.

MultiCluster Enhancements

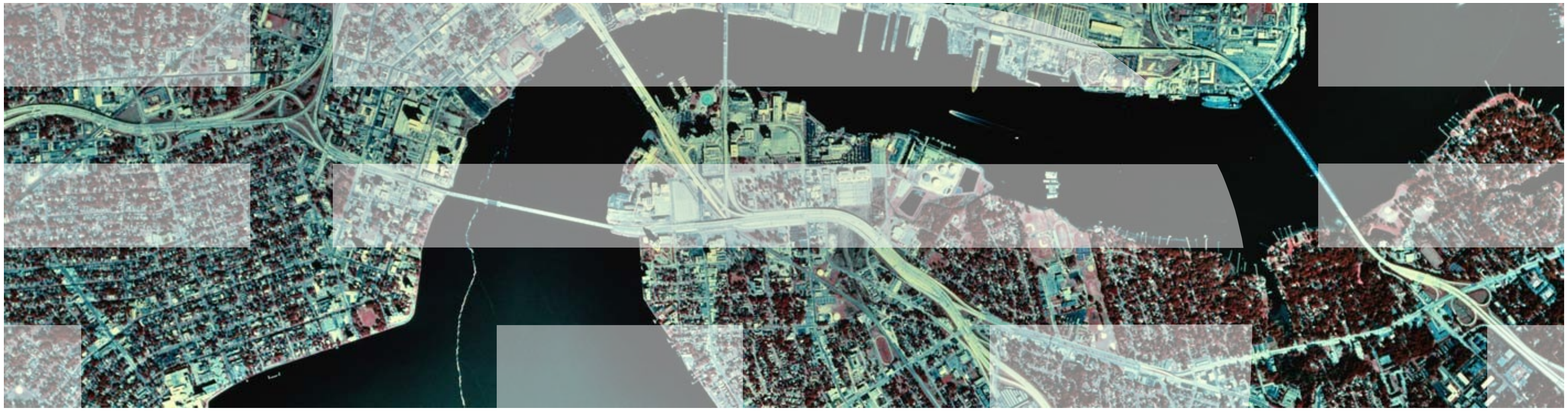
- Dynamic Cluster Weighting
  - A new job forwarding cluster load balancing policy to consider cluster preference, available slots, pending queue length and slot threshold on remote queues.

- Numeric and String Resource Aware Scheduling
  - LSF Multi-cluster scheduler has been enhanced to recognize numeric and string resources during job forward resource evaluation (select clause)

- Multi-site configuration management.
  - Support #include syntax in lsf.shared and lsb.applications to allow clusters sharing the common configuration across multiple sites.

  - Support inconsistent lsf.shared files across multiple clusters
  - Allow sites have different resources definitions in its own lsf.shared file.

IBM Platform LSF 9.1.1
1Q 2013

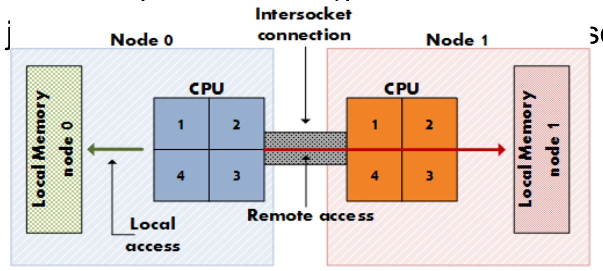


Scheduling: CPU and Memory Affinity (LSF 9.1.1)

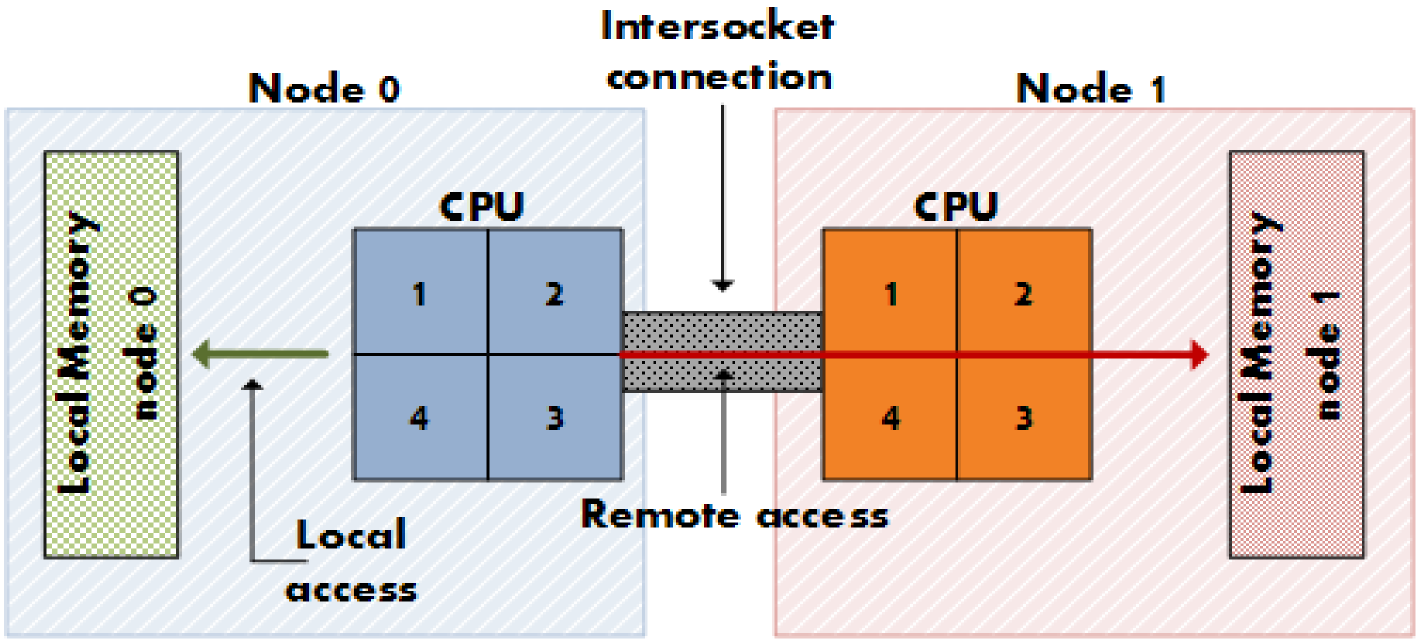
- Allocating cores and memory is sometimes insufficient to ensure efficient use of hosts
- Due to the NUMA nature of today’s multicore hosts, it is possible that the internal processor memory bandwidth can be saturated well before all the cores are allocated.

- LSF has supported complex core/memory placement on large systems such as SGI Origin/Altix/UV for 10+ years – but this is too heavyweight for dual socket nodes.

- LSF v7u1 introduced a simple interface to bind processes to cores/sockets
  - Simple user interface:
    - Hides all apid ordering weirdness from the user
    - BIND_JOB parameter in application profile with following values: ANY|PACK|BALANCE|USER
  - Constrains:
    - Limited to single node jobs
      - Binding for multi-host jobs was delegated to the MPI layer
    - Does not consider j socket or NUMA node and no memory affinity setting



Scheduling: CPU and Memory Affinity (LSF 9.1.1)



Affinity Requirement

- Syntax
  - Processor unit type and number per slot: ***"type(num)"***. Type can be "numa", "socket", "core", "thread". i.e, "affinity[core(1)]".
  - Memory binding option*: **"membind=localonly | localprefer"***
  - Cpu binding level: ***"cpubind=type"***. Type can be "numa", "socket", "core" and "thread". The default will be same as per slot processor unit type
  - Task distribution: ***"distribute=pack|pack(type=1)|balance|any"***.

- Examples
  - *-n 6 –R "span[hosts=1] affinity[**core**(1):**distribute**=pack]"*

  Job asks for 6 slots, runs within single host. Each slot maps to one core and try to pack 6 cores as close as possible on single NUMA or socket. If cannot be done, job can still run.

    - *-n 6 –R "span[hosts=1] affinity[**core**(1):**distribute**=pack(socket=1)]"*

  Job asks for 6 slots, runs within single host. Each slot maps to one core and **MUST pack into single socket, otherwise, job need to pend**.

Advanced Syntax

- One slot/task requires multiple segments of processor units and each segment has special topology requirement.
  - Useful in HPC OpenMP environment.
- Syntax
  - *affinity[**type**(num, **same**=same_level, **exclusive**=(excl_level,excl_scope)) * count]*
  - **same_level** can be numa, socket, core, must be higher than “type”.
  - **Excl_level** can be numa, socket, core.
  - **Excl_scope** can be intask, injob, alljobs.

  - Example
    - *-n 2 –R “span[hosts=1] affinity[**core**(2, same=socket, exclusive=(socket, injob)]”*

Job asks for 2 slots on a single host. Each slot maps to 2 cores. 2 cores for a single slot/task must come from the “same socket”, however, the other 2 cores for second slot/task must be on different socket.

IBM Platform LSF Add-on Products

New Project Mode

- **V8 introduced a new scheduling model which addressed many customer pain points and rekindled interest**
- **V9 principally focuses on addressing edge cases that were in use by existing V7 customers, but not supported by the V8 model**

More flexible hierarchal project definition

Better handling of parallel jobs where each rank checks out a license directly

Removal of the need for scheduler restarts when license scheduler configuration changes

Alert the admin when LSF and License Scheduler cannot communicate.

Integration with IBM Parallel Environment

Increased Scalability ~100K ranks

Oversubscription mode for testing

MPI3.0 Preview: Non blocking collectives, HA Support

Enhanced Collective algorithms

TCP & IB On Demand Connections

IB to TCP Failover

Checkpoint/Restart Preview – “super barrier”

VM Testing

CUDA & FCA Update

Intel MIC Support

IBM Platform LSF Add-on Products

Application Center 9.1

Configurable information retention period

"badmin perfmon" metrics to be available in the dashboard

Additional columns

- Job runtime information –WF –WL –WP
- Job Description –Jd
- Custom Column content

RBAC on job details

WS-API – Extended Job Status, additional functionality: requeue, signal, migrate

Additional support for 3D visualisation technologies (DCV, EoD, Citrix)

Template extensions:

- Custom Help within Templates
- Handling of 'include' files when uploading.

Improved interface between IBM Platform Application Center and IBM Platform Analytics

Process Manager 9.1

Support for non LSF batch systems

Support for IBM Platform Symphony

Support for user variables when running flows spanning multiple clusters.

Note: There were several interim releases of Platform Process Manager 8 (8.0.1, 8.0.2, 8.0.3) which were rolled up into IBM Platform Process Manager 8.3

The delta between 8.3 and 9.1 are the final set of projects in the original scope

IBM Platform LSF Add-on Products

Platform RTM 9.1

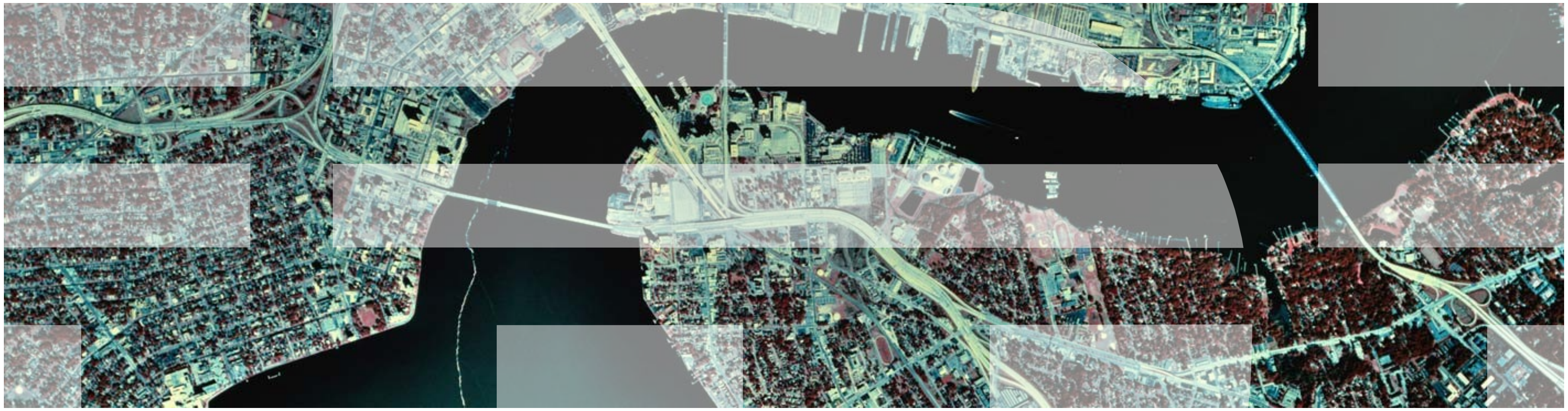
Support for Scientific Linux
ELIM Templates
Support for GPFS Monitoring
Graphing of LSF GSLA, Dynamic Cluster
Refresh of the Alarms module
Enhanced Filtering and Drilldown capabilities

Platform Analytics 9.1

For LSF:
- FlexLM License Reporting
- FlexNet Manager Reporting
- License Denial Reporting
- Pending Reason Analysis
- Plus updates to existing workbooks
- Enhanced integration with Application Center

Support for IBM Platform Symphony

IBM Platform LSF Additional Features



Preemption

- **Configuration to modify preemptive scheduling behavior**
  - There are configuration parameters that modify various aspects of preemptive scheduling behavior, by
  - Modifying the selection of the queue to preempt jobs from
  - Modifying the selection of the job to preempt
  - Modifying preemption of backfill and exclusive jobs
  - Modifying the way job slot limits are calculated
  - Modifying the number of jobs to preempt for a parallel job
  - Modifying the control action applied to preempted jobs
  - Control how many times a job can be preempted
  - Specify a grace period before preemption to improve cluster performance

Blaunch

- **blaunch**
  - launches parallel tasks on a set of hosts

- **Synopsis**

```
blaunch [-n] [-u host_file | -z host_name ... | host_name] [-use-login-shell |
-no-shell ] command [argument ... ]
blaunch [-h | -V]
```

Blaunch

- **Description**
  - Important: You cannot run blaunch directly from the command line.
  - Restriction: The command blaunch does not work with user account mapping. Do not run blaunch on a user account mapping host.
  - Most MPI implementations and many distributed applications use rsh and ssh as their task launching mechanism. The blaunch command provides a drop-in replacement for rsh and ssh as a transparent method for launching parallel applications within LSF.
  - blaunch supports the following core command line options as rsh and ssh:
  - rsh *host_name command*
  - ssh *host_name command*
  - All other rsh and ssh options are silently ignored.
  - blaunch transparently connects directly to the RES/SBD on the remote host, and subsequently creates and tracks the remote tasks, and provides the connection back to LSF. You do not need to insert pam, taskstarter or any other wrapper.

Blaunch

- **Options**

-n Standard input is taken from /dev/null. (Not supported on Windows.)

-u *host_file* Executes the task on all hosts listed in the *host_file*.

Specify the path to a file that contains a list of host names. Each host name must listed on a separator line in the host list file.

This option is exclusive of the -z option.

host_name The name of the host where remote tasks are to be launched.

-z *host_name* ... Executes the task on all specified hosts.

Whereas the host name value for rsh and ssh is a single host name, you can use the -z option to specify a space-delimited list of hosts where tasks are started in parallel.

Specify a list of hosts on which to execute the task. If multiple host names are specified, the host names must be enclosed by quotation marks (" or ') and separated by white space.

This option is exclusive of the -u option.

-use-login-shell Launches commands through user's login shell.

Only applies to UNIX and Linux hosts.

-no-shell Launches commands without any intermediate shell.

command [*argument* ...] Specify the command to execute. This must be the last argument on the command line.

-h Prints command usage to stderr and exits.

-V Prints LSF release version to stderr and exits.