Job Scheduling in HPC systems

Julita Corbalan
EEAP

Outline

• Definitions
• Architectures
• Scheduling
  – Variable
  – Dynamic
  – Two-levels
  – Meta-scheduling
• Performance evaluation
• Some existing queuing systems

Definitions

• Job
  – Unit to schedule. Includes a description of the work (executable, parameters, etc) and a list of resource requirements (number of cpus, amount of memory, etc)
• Resource manager
  – Component of the system that performs the low level process management. Typically includes node selection.
• Job scheduler
  – Component of the system that selects the next job to be executed from a list of queued jobs
Definitions

• Jobs can be classified in Rigid, Moldable, or malleable, based on the capacity to adapt their number of processes
  — Rigid
    • Number of cpus is fixed and determined at submission time
  — Moldable
    • Number of cpus is fixed and determined at start time
  — Malleable
    • Number of cpus can be changed at runtime

Architectures

• SMP’s
  • Shared memory
  • Global system view
• Clusters of workstations
  • N local workstations (1 cpu)
  • Distributed memory
• Clusters of SMPs
  • Two levels: Local and Global
  • Big systems
• BIG Shared memory machines
  • Less than clusters
  • Global system view
• GRIDs
  • N administrative domains
  • Low coupled systems
HPC Arquitectures
(http://www.top500.org/)

- 31 racks → 6 Blade centers
- Each blade centers → 14 blade js21
- Each JS21
  - 2 PowerPC 970MP at 2.3 GHz
  - 8 Gb of shared memory
  - local SAS disk of 36 Gb.
- Each Power pc
  - 64 bits
  - superscalar processor with vectorial extensions type SIMD
- 10240 processors!

HPC system architectures

Resource Manager
Resource Manager

• Low level process management
• Spawn processes
• Monitor processes
• Check job status
• Implements node selection policies

• Usually not considered in job scheduling research work
  – Even though it is important [Guim07]

Resource selection: SLURM

• Linear
  • Selects nodes assuming a one-dimensional array of nodes.
  • The nodes are selected so as to minimize the number of consecutive sets of nodes utilizing a best-fit algorithm
  • Does not allocate individual processors, memory, etc.

• cons_res
  • Can allocate individual processors, memory, etc. within nodes
  • Recommended for systems with many non-parallel programs sharing nodes

• Bluegene
  • Node selection specific for blue gene

Resource selection: TORQUE

• Not explicit policies
• Based on resource requirements
• Allows external scheduler
• First fit
Resource selection: Load Leveler

- Based on resource requirements
- Allows external scheduler
- First Fit

Job scheduling

- Time-sharing
- Space-sharing
  - Variable
  - Dynamic partitioning
- Mixed (Two-levels)
  - Gang scheduling
  - Co-scheduling
Time sharing

- Processors schedules processes uncoordinatedly
- Local queues vs. Global queues
- Not useful “as is” for HPC applications, only combined with space-sharing
  - Gang scheduling
  - Co-scheduling

Space-sharing

- The machine is partitioned into disjoint parts
- Jobs are executed into partitions
  - Typically in a Run to completion way
- Historical approaches
  - Fixed: hardcoded
  - Variable: Can change but not at runtime
  - Dynamic: Can change at runtime

Variable partitioning

FCFS
Backfilling: runtime estimation
Space-sharing

Typically the system is partitioned into queues with different sizes and limits
- Number of cpus
- Execution time
- Type of users

Schedule jobs submitted to each queue with scheduling policy
- FCFS
- Bsckfilling based policies

FCFS

Activated at job arrival and job ends
- If there are enough free processors to start the FIRST job in the queue → start it
- Else wait

Advantage:
- Very simple to implement
- Fairness

Disadvantages:
- Fragmentation: The first job blocks the rest of the queue
Job scheduling for HPC systems

Julita Corbalan

Time

Processors

Job1

Job2

Job3

Job4

Backfilling strategies

• Idea
  • Backfill jobs to the head of the queue, break the FCFS order

• Goal
  • Fill holes in the systems

• Restrictions
  • Defines N reserved jobs
  • Backfilled jobs can not delay reserved ones

• Requirements
  • Job runtime estimations

• How to reduce the fragmentation?
  • Breaking the strict arrival order, allowing the execution of jobs that have arrived later
  • Allow the possibility of introducing priorities in the queue
    – Backfilling is not a policy, is a family of policies
  • Skill Run-To-Completion

Backfilling

FCFS

Job Scheduling for HPC systems

Julita Corbalan

Job1

Job2

Job3

Job4

Processors

Time

03/12/2007
Backfilling strategies

EASY backfilling

- Queue is ordered in FCFS order
- One reservation
  - The scheduler only guarantees that the FIRST job in the queue is not delayed
- Conditions to backfill a job:
  - Its (estimated) runtime is less than the start time of the first job (shadow time)
  - It requests less processors than the "extra processors"
    - Processors that will remain free when the first job starts.

First Case
First Case

Second Case

Backfilling Properties

• Unbounded delay
  – Backfill jobs will not delay first queued job
  – But they may delay other queued jobs...
Delay (job 2 is reserved)

Job Scheduling for HPC systems
Julita Corbalan

Delay

Job Scheduling for HPC systems
Julita Corbalan

Backfilling strategies

• How to organize the queue?
  – FCFS
  – Priorities
• How many reservations?
  – One reservation
  – N reservations
  – Dynamic
  – etc
• How many jobs are analyzed to backfill?
  – One job
  – N jobs
• What action to take if runtime estimation is incorrect?
  – Kill job
  – Delay the reserved job!!!
Other backfilling proposals

- Short the queue by runtime
  - SJF-Backfilling
- Use two queues
  - FCFS queue for global queue
  - SJF queue for backfilled jobs

Using prediction for scheduling

- Backfilling is based on user runtime estimations
- Estimates are expected to be accurate
  - Small (and accurate) estimations allow short jobs to be backfilled in the queue
- However, too short estimations can result in killing the job 😞
Run-time estimations

• Lots of Works have analyzed the user behavior
• Users tends to overestimate their jobs
• Few values are used to estimate all the jobs runtimes
  — 20 values for the 90% of the jobs
  — Queue limits
• Modeling user runtime estimates
  •  [http://www.cs.huji.ac.il/~dants/papers/Est05JSSPP.pdf](http://www.cs.huji.ac.il/~dants/papers/Est05JSSPP.pdf)

Runtime estimations/predictions

• Some surprising result shows that inaccurate estimations doesn’t suppose worse performance metrics
  • Backfilling is influence by the evaluation metrics and some errors tend to generate sjf policies
• The Dynamics of backfilling
  •  [http://www.cs.huji.ac.il/~feit/papers/BfDyn06IISWC.pdf](http://www.cs.huji.ac.il/~feit/papers/BfDyn06IISWC.pdf)

Runtime prediction

• Rather than using user estimation... use prediction
  — Easy for the user
  — Quite accurate
  — However, what happens ...
  • If no info is available
  • When a miss prediction happens: the job can not be killed
  •  [http://www.cs.huji.ac.il/~feit/papers/BfDyn06IISWC.pdf](http://www.cs.huji.ac.il/~feit/papers/BfDyn06IISWC.pdf)
Runtime prediction

- Several works concerning prediction have been already published:
  - Mathematical and statistical
    - P. Dinda
      - Based on Time Series for prediction CPU load
      - Based on Time Series for host load prediction.
    - Yuanyuan based on the work of P. Dinda uses also Time Series
      - M. V. Devurakonda based on transition diagrams and linear regressions
    - Allen B. Downey uses statistical approaches
  - Data mining techniques
    - Gibbons static templates using and linear
    - Hui Li K-Nearest-Neighbour
    - Warren Smith dynamic templates (using genetic algorithms)

Dynamic partitioning

- Partitions are modified based on jobs queued and jobs running
- Requires moldable/malleable jobs
  - Not so easy to support
- Policies:
  - Equipartition
  - Performance-Driven
  - Equal efficiency
  - Proportional
  - ...
Dynamic partitioning

• Equipartition [McCann96]
  – Applied at job arrival and job finish
  – Each application receives the same amount of cpus

\[ cpus = \min\left\{ \frac{\text{Total cpus}}{\text{number of jobs}} \right\} \]

• Performance-Driven [Corbalan00]
  – Coordinated with a long-term scheduler
    • Running FCFS, Backfilling, etc
  – Jobs receive more or less processors based on the reached speedup and the requested number of processors
  – Speedup is dynamically measured by a RTL
  – Policy is a search algorithm: implements a state diagram

• Equal – Efficiency [Nguyen 96]
  – Applied at each job arrival and job ends
  – Efficiency of applications is measure at runtime
    • The scheduler has a plot per job
  – Scheduler distributes processors in a way that all the running jobs reach the same efficiency
  – However, authors assume SAME efficiency means GOOD efficiency, and this is not true
Dynamic Partitioning

- Proportional
  - Applied at job arrival and job finish
  - Each job receives an amount of cpus proportional to the number of cpus requested

Dynamic partitioning

- Moldable
  - PSA (Processor Saving Adaptive):
    - Takes into account queued jobs. Equipartition available processors
  - ASP (Adaptive Static Partitioning)
    - Allocates a percentage of available processors, with a Maximum, i.e. ASP-60 is 60% maximum of the total pool of processors

Folding

- Malleable
  - Folding
    - Folds oldest application to free processors, processes time share processors
Two-level scheduling

Gang scheduling
Co-scheduling

Mixed (two-level) scheduler

- The system is partitioned
- Time-sharing is performed in these paritions
- Approaches:
  - Gang scheduling
  - Co-scheduling

Gang scheduling

- Like backfilling, it is a family of strategies
- The system is organized like a table
  - Columns are time-slices (quantum)
  - Rows are processors
- Processes of applications are mapped one-to-one to processors
- Each quantum, all the processors switch to next column
- The set of processes of a job is called a gang
Gang scheduling

66% of the system is used vs. 100% utilization

P0
P1
P2
P3
P4
P5
P6
P7

Gang scheduling

- Advantages
  - Reduces fragmentation
  - Reduces the impact of incorrect scheduling decisions

- But
  - Overhead for context switch & coordination
  - High memory pressure (and resources in general)
  - Reduces cache efficiency

Gang scheduling parameters

- Packing algorithm
  - Once a new job arrives, how do I put it in the table?
  - Once a job finished, Is the table reorganized?
  - Having an efficient packing algorithm is critical for gang scheduling performance

- Quantum???
Co-scheduling

• Based on the same concept than Gang scheduling
  – All the processes of a parallel job must be executed at the same time
• However, it relaxes this requirement
• Tries to execute as many processes as possible of the same job at the same time

Co-scheduling

• Gang scheduling is Explicit:
• Co-scheduling is Implicit:
  – Local decisions are based on communication events
    • How to wait for a message
      – spin block, periodic boost
    • What to do on message arrival
      – periodic boost
    • Targeted to clusters

Flexibility

• DCS: Demand-based co-scheduling
• ICS: Implicit co-scheduling
• FCS: Flexible co-scheduling
Demand-based Coscheduling

• Prioritize communicating threads
• Switch to a thread that receives a message
  – Only if that thread has received less than its fair-share of cpu. Prevents monopolizing the system.
  – Also defines "epochs". When a node switches a new epoch starts.

Implicit co-scheduling

• Threads waiting for messages gain priority
• Use two-phase blocking to ensure sending threads stay scheduled
  – Waits for 5 context-switch durations

Flexible co-scheduling

• Mix of co-scheduling and Gang scheduling
• Dynamic classification of jobs
  – If communicate a lot \( \rightarrow \) try to co-schedule at the same time
  – Else \( \rightarrow \) not needed to co-schedule
Co-scheduling

<table>
<thead>
<tr>
<th>Waiting for a message</th>
<th>Message arrival</th>
<th>Busy wait</th>
<th>Spin block</th>
<th>Spin Yield</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Explicit Re-Schedule</td>
<td>Local</td>
<td>Spin Block</td>
<td>Spin Yield</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt &amp; Re-Schedule</td>
<td>Dynamic Co-Scheduling</td>
<td>Dynamic Co-Scheduling - Spin Block</td>
<td>Dynamic Co-Scheduling - Spin Yield</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Periodically Re-Schedule</td>
<td>Periodic Boost</td>
<td>Periodic Boost - Spin Block</td>
<td>Periodic Boost - Spin Yield</td>
<td></td>
</tr>
</tbody>
</table>

Meta-scheduling scenarios

Related work: Global Dispatcher

- Job queues
  - At the dispatcher level
  - At local level
- Scheduling
  - One centralized scheduling policy
  - Local policies are disabled
  - Local resource managers (MAUI, SLURM etc.)
- User interacts with the global dispatcher
Related work: Global Scheduling Optimizer

- Job queues
  - At local level
- Scheduling
  - Local Schedulers (MAUI, SLURM etc.)
  - Local resource managers (LoadLeveler, OpenPBS etc.)
  - Global Component
    - Scans the job queues and decides if they can start earlier in another resource
- User interacts with the local schedulers

Related work: Global Scheduler

- Job queues
  - At the dispatcher level
  - At local level
- Scheduling
  - One centralized global scheduling policy (Broker type)
  - Local Schedulers
  - Local resource managers
- User interacts with the global scheduling

Related work: Pull Model

- Job queues
  - At the global level
- Scheduling
  - Local Schedulers pull jobs from the centralized queue
  - Local resource managers
- User pushes the jobs to the global queue
Related work: AppLess

- Job queues
  - Local queues
- Scheduling
  - The application AppLess Agent schedules the tasks of the application based on:
    - Resource status
    - Task requirements
  - Local resource managers
  - Middleware:
    - Legion
    - Globus
- User interacts with its AppLess Agent

Task Assignment Policies

- Where the tasks should be submitted:
  - Random (tasks are uniformly distributed)
  - Less-SubmittedJobs (round robin submission)
  - Shorts-Queue (the resource with less queued jobs)
  - Shorts-Queue (the resource with less pending work)
  - SITA-E (task is submitted to the resource based on its runtime)
  - SITA-U-opt (minimize the mean slowdown)
  - SITA-U-fair (balance the slowdown for large jobs equal to short jobs)
  - Less-WT, Less-SDL, [Guim07]

Job dispatching

- Less-WaitTime, Less-*
  - Jobs are dispatched based on:
    - Jobs requirements: cpus, etc
    - Resource metrics: Estimated wait time, etc
    - Runtime prediction
  - ISIS-Dispatcher
  - Job-guided scheduler
  - Negotiates with HPC centers
  - No global scheduler
  - Local policies and submissions are supported
  - Local schedulers has to extend their API
Workloads

- Basic for job scheduling evaluation
- Describes the set of jobs to be executed
- It must be representative
- Standard Workload Format (SWF) proposed by Dror Feitelson
- Useful for workload trace files and workload models

SWF

- Header file: describes system characteristics where the trace file was collected
- Jobs list:
  - Job Number, Submit Time, Wait Time, Run Time (The wall clock time the job was running)
  - Number of Allocated Processors, Average CPU Time Used (both user and system, in seconds)
  - Used Memory, Requested Number of Processors, Requested Time, Requested Memory
  - Status
  - User ID, Group ID
  - Executable (Application) Number
  - Queue Number
  - Partition Number
  - Preceding Job Number
  - Think Time from Preceding Job
Jobs

- **Rigid**
  - Number of cpus is fixed and determined at submission time

- **Moldable**
  - Number of cpus is fixed and determined at start time

- **Malleable**
  - Number of cpus can be changed at runtime

Job Scheduling for HPC systems

Julita Corbalan

Workload trace files

http://www.cs.huji.ac.il/~jcorbalan/workload/logs.html

Job Scheduling for HPC systems
Julita Corbalan
Workload trace files

• Logs the jobs (and characteristics) submitted in a HPC center for several months
  – Theoretically it is representative
  – However, it is influenced by system configuration
    • “Long jobs are submitted and executed at night…”
    • But...a lot of HPC centers only allow big queues at night
  • Characteristics of submitted jobs depends on features supported in the center

Workload Trace file

• Results highly depend on the trace file used
  – You MUST use more than one
  – You MUST know workload characteristics
• Characteristics evolves a lot

Workload trace files

• Cleaned versions
  – Anomalies are detected and removed when converting original log format for SWF format
  – Recommended
• Flurries
  – Burst of very high activity by a single user
  – They influence in performance evaluation
  – Filters are provided to remove them
Job Scheduling for HPC systems

Julita Corbalan

Workload models
http://www.cs.huji.ac.il/labs/parallel/workload/models.html
Workload models

• Downey 97
  – Modeling of job runtimes
  – Modeling moldable jobs
  – Code is provided
• Cirne and Berman 2001
  – A model for generating a stream of rigid jobs
  – A model for turning the rigid jobs into moldable ones
  – Code is provided
• Tsafrir, 2005
  – Realistic distribution of user runtime estimates
  – Code is provided
  – Generates n jobs or adds information to existing SWF file

Analysis of workloads

• http://www.cs.huji.ac.il/labs/parallel/workload/wlbib.html

Evaluation considerations
Performance metrics

- Wait Time
- SLD/BSLD/Weighted SLD
- Backfilled Jobs
- Processors used per interval of time
- Killed jobs
- Etc...

Job Performance metrics

- Wait time
  - Does not take into account the runtime of the job
  - All the jobs have the same weight
- Slowdown
  - It is a function of the runtime of the job
  - Very influenced by short jobs
- BSLD
  - Reduces the influence of short jobs by setting a minimum (60 sec.-5 min.)

Job Performance metrics

- Weighted slowdown
  - Each job has a weight
  - Weight is the fraction of the job in the total of the workload
  - Minimizes the impact of short/small jobs

\[ W = SLD (j) = SLD (j) \times Weight (j) \]
Slowdown

System metrics

- Backfilled Jobs
- Processors used per interval of time
- Killed jobs

Statistics estimators

- Very important in order to take conclusions
  - Mean
  - Stddev
  - Percentile (95%)
  - Distribution Normal/Uniform
Evaluation techniques

- Real executions
  - Disadvantages
    • Difficult to execute representative workloads
    • Difficult to change parameters
  - Advantages
    • Includes all the runtime considerations
    • Real evaluations

Evaluation techniques

- Simulations
  - Advantages
    • High number of jobs can be simulated
    • Easy to test new configurations
    • Easy to repeat by other researchers if well defined
  - Disadvantages
    • Usually only models part of the system
    • Very difficult to validate
    • It is difficult to define the impact of non considered factors
Existing queueing systems

Load Leveler: Architecture
1. Submission
2. Schedd contacts with central manager
3. Scheduling!
4. Shadow process creates startd process

LoadLeveler API
http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IBMp690/IBM/usr/lpp/LoadL/html/am2ugmst58.html
- Accounting API
- Error Handling API
- Checkpointing API
- Submit API
- Data Access API
- Parallel Job API
- Workload Management API
- Query API
- User exits
LoadLeveler API: Workload management

- **ll_control subroutine**
  - This subroutine allows an application program to perform most of the functions that are currently available through the standalone commands: llct, llfavorjob, llfavoruser, llhold, and llprio.

- **ll_modify subroutine**

- **ll_preempt subroutine**

- **ll_start_job subroutine**
  - This subroutine tells the LoadLeveler negotiator to start a job on the specified nodes

- **ll_terminate_job subroutine**
  - This subroutine tells the negotiator to cancel the specified job step.

MAUI/MOAB


- MAUI & MOAB are Schedulers
  - They do not manage directly resources
  - Tells the resource manager what to do, when to run jobs, and where.

MOAB scheduler
SLURM

http://www.llnl.gov/linux/slurm/

- Simple Linux Utility for Resource Management
- Open Source product
- Main features
  - Allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work
  - Provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes
  - Arbitrates conflicting requests for resources by managing a queue of pending work

SLURM commands

- Sbatch is used to submit a job script for later execution.
- Srun is used to submit a job for execution or initiate job steps in real time.
- Scancel is used to cancel a pending or running job or job step.
- Squeue reports the state of jobs or job steps
- Etc...

SLURM architecture
SLURM plugins

- General-purpose plugin mechanism available to easily support various infrastructures
  - Authentication of communications
  - Checkpoint
  - Cryptography
  - Job Accounting Gather
  - Job Accounting Storage
  - Job completion logging
  - MPI
  - Node selection
  - Process tracking for signaling
  - Scheduler (The Maui Scheduler, Moab Cluster Suite, backfill, or FIFO (default))
  - Switch or interconnect

SLURM plugins: scheduler

- different models: builtin, backfill, wiki and wiki2
- API
  - int slurm_sched_plugin_reconfig (void);
  - int slurm_sched_plugin_schedule (void);
  - uint32_t slurm_sched_plugin_initial_priority (uint32_t last_prio, struct job_record *job_ptr);
  - void slurm_sched_plugin_job_is_pending (void);
  - void slurm_sched_plugin_partition_change (void);
  - int slurm_sched_get_errno (void);
  - const char *slurm_sched_strerror(int errnum);

SLURM plugins: node selection

- Models: linear, cons_res, blue-gene
- API
  - int select_p_state_save (char *dir_name);
  - int select_p_state_restore (char *dir_name);
  - int select_p_node_init (struct node_record *node_ptr, int node_cnt);
  - int select_p_block_init (List block_list);
  - int select_p_update_block (update_part_msg_t *part_desc_ptr);
  - int select_p_pack_node_info (time_t last_query_time, Buf *buffer_ptr);
  - Etc...