Moab Evaluation

Projekt Parallelrechnerevaluation

Florian Ehmke
8ehmke@informatik.uni-hamburg.de

University of Hamburg

1. Juli 2011

Index

Introduction / Motivation

Energy aware High-performance computing

- Often the energy costs are a large amount of the total costs of ownership (TCO) of a cluster
- Even with current Energy-saving mechanisms the power consumption during idle phases remains high
- Shutting down idle nodes seems like a viable option

Introduction / Motivation

Energy aware High-performance computing

- 5 AMD nodes (each 2 Opteron 6168 CPUs)
  - 5x 2x 12 cores @1,9 GHz
- 5 Intel nodes (each 2 Xeon Nehalem X5560 CPUs)
  - 5x 2x 4 cores (8 Threads) @2,8 GHz

Purpose of this project

- Evaluate Moab's green features on the eeClust
- Analyse workload on the Blizzard cluster

Scheduling / Resource management

General

Resource manager

A resource manager monitors the state of nodes (power, load, jobs etc.), receives new jobs and executes them on compute nodes.

Scheduler

The scheduler tells the resource manager what to do (when and where to run jobs).

Current installation on eeClust

- Resource manager: `Torque`
- Scheduler: `Maui`

Scheduling / Resource management

Moab

What normally happens when a node is shutdown:

1. Resource manager detects that node is off
2. Resource manager tells scheduler that node is off
3. Scheduler (e.g. `Maui`) marks node as down and completely ignores that node for future scheduling

Scheduling / Resource management

What is Moab?

- Powerful resource management and scheduling system
- Works with other resource management and monitors (`Torque, PBS, IPMI, Ganglia` etc.)
- Closed source, basic functions similar to `Maui`
- Moab, `Maui`, `Torque` developed by Adaptive Computing[1]
- Supports shutting down idle nodes (without disturbing the scheduling)

[1] `http://www.adaptivecomputing.com/`

Scheduling / Resource management

Moab

What happens when a node is shutdown by Moab:

1. Moab initiates shutdown (e.g. via `IPMI`)
2. Resource manager detects that node is off
3. Moab continues to regard the node as available and uses its resources for scheduling
4. If any job needs the resources of that node Moab boots that node again and waits for the resource manager
5. As soon as the resource manager detects the node as available Moab submits the job

Moab Setup

Installation

1. `./configure`
2. `./make install`

Configuration - Server

```
RMCFG[Moab] TYPE=PBS
RMCFG[Moab] SUBMITCMD=/usr/bin/qsub
RMCFG[Moab] SBINDIR=/usr/sbin
```

Configuration - Nodes

```
SCHEDCFG[Moab] SERVER=eeclust:42600
```

Moab Setup

Configuration - Green features

```
NODECFG[DEFAULT]               POWERPOLICY=OnDemand
MAXGREENSTANDBYPOOLSIZE        0
NODEIDLEPOWERTHRESHOLD         150

RMCFG[intel]      TYPE=NATIVE
RMCFG[intel]      CLUSTERQUERYURL=exec:///query.py
RMCFG[intel]      NODEPOWERURL=exec:///power.py

PARCFG[intel]     NODEPOWERONDURATION=1:10
PARCFG[intel]     NODEPOWEROFFDURATION=0:10
```

eeClust

Energy Saving Potential

	T_{boot}	E_{boot}	$T_{shutdown}$	$E_{shutdown}$	P_{idle}	P_{off}	T_{min}
intel	70 s	10,5 kJ	10 s	1,2 kJ	133 W	8 W	89.12 s
amd	70 s	12,25 kJ	30 s	3,6 kJ	105 W	8 W	155.15 s

Table: Duration of boot and shutdown, Energy consumption during boot and shutdown, Power consumption during idle and off and min. idle time

How to calculate T_{min} (Break-even point)

$$T_{min} = \frac{P_{off} \times T_{boot} + P_{off} \times T_{shutdown} - E_{boot} - E_{shutdown}}{P_{off} - P_{idle}}$$

eeClust

Scenario

- All jobs execute `partdiff-par`
- Jobs run between 5 and 20 minutes
- Different backfilling algorithms were applied:

`FIRSTFIT` The first job that fits into the current backfill window will be started.

`BESTFIT` For each job that fits into the current backfill window a *degree of fit* will be calculated. The Job with the best *degree of fit* will be started.

`GREEDY` A *degree of fit* for all possible combinations of jobs that fit into the backfill window will be calculated. The best combination will be started.

eeClust

Measurements

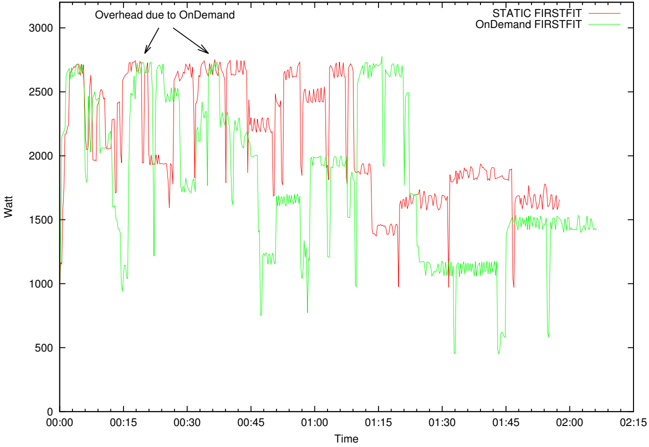

Figure: Process of the power consumption (STATIC vs. OnDemand)

eeClust

Measurements

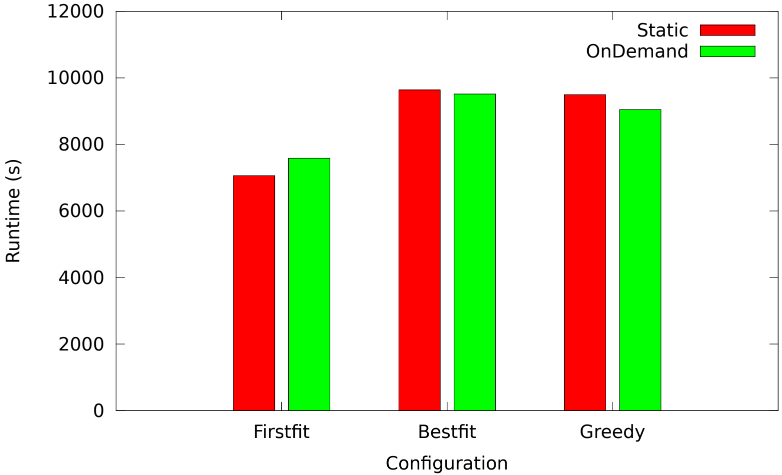

Figure: Total time elapsed in seconds during the 6 different test runs

eeClust

Measurements

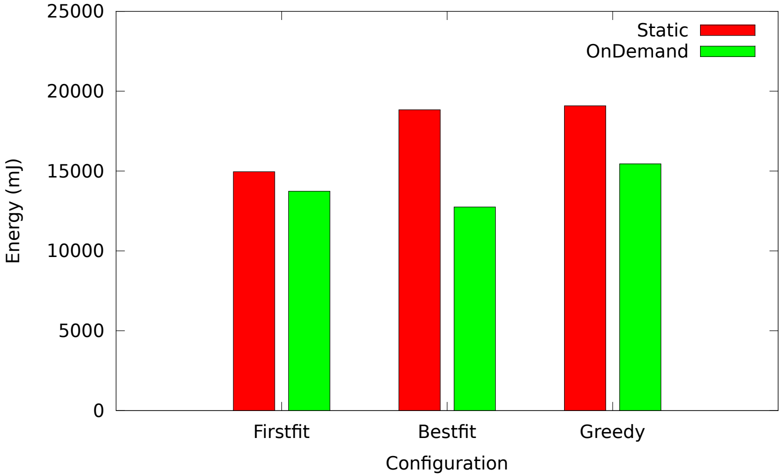

Figure: Total energy consumption in mJ of each different configuration

eeClust

Measurements - Energy-delay-product

- Saving energy does not justify **very** long runtimes
- Energy-delay-product gives an idea how good/bad the relation between the consumed energy and the runtime is
- EDP is defined as follows:

$$EDP = Joule \cdot seconds$$

eeClust

Measurements

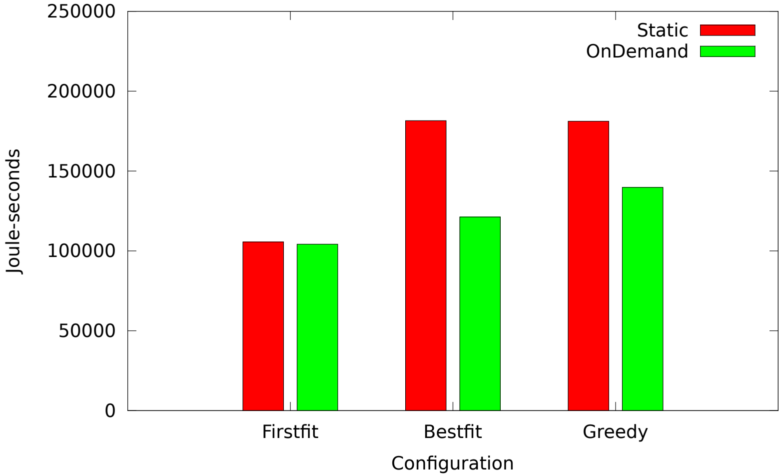

Figure: Energy-Delay-Product of the 6 configurations (smaller is better)

eeClust

Measurements

- `OnDemand` `POWERPOLICY` results in noticeable overhead
- It saves enough energy to achieve better EDP score
- Scheduling changes due to the overhead

Blizzard

Hardware

- computational power: 158 TeraFlops/s
- 264 IBM Power6 nodes
- 16 Dual-core CPUs per node (8448 cores total)
- more than 20 TeraByte main memory
- Infiniband network

Blizzard

Energy Saving Potential

state	duration	Power Consumption
boot	15,5 - 30 min	2550 W - 4250 W
shutdown	5 - 6 min	2550 W - 4250 W
idle	-	2550 W - 3083 W
off	-	ca. 100 W

Table: Power Consumption during boot and shutdown

min. time (Break-even point)

worst case: 3659 s (ca. 61 min)
best case: 2083 s (ca. 35 min)

Blizzard

Load investigation

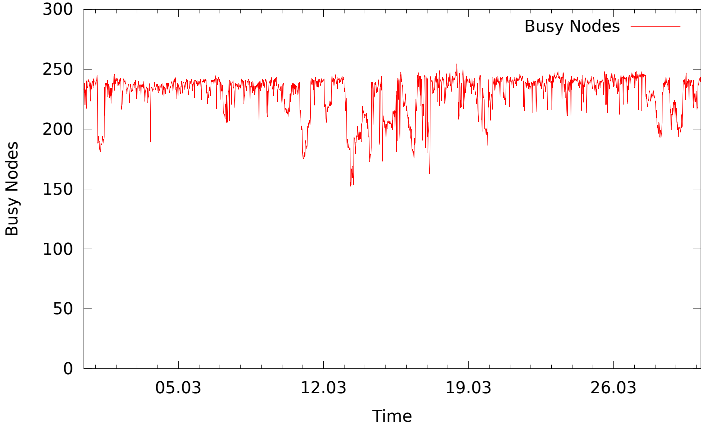

Figure: Load over one month (May 2011) on the DKRZ Blizzard cluster

Blizzard

Load investigation

- Overall available CPU hours: 5668949
- CPU hours spent idle: 339365 (5.986%)
- 286025 could have been spent shutdown (5.045%)
- Possibly saved kilowatt hours (kWh): 22793
- Possibly saved money (0,13€ per kWh): 2963 €

Conclusion

- Shutting down one node to save energy is a drastic action
- Savings depend on the workload for the most part
- OnDemand increases runtime and changes scheduling
- Saved energy evens out runtime overhead (EDP)

Thank you for your attention.