Scheduling of Multicore Jobs

Nathalie Rauschmayr

LHCb collaboration

May 20, 2014

Outline

Introduction

Going rapidly towards many core systems:

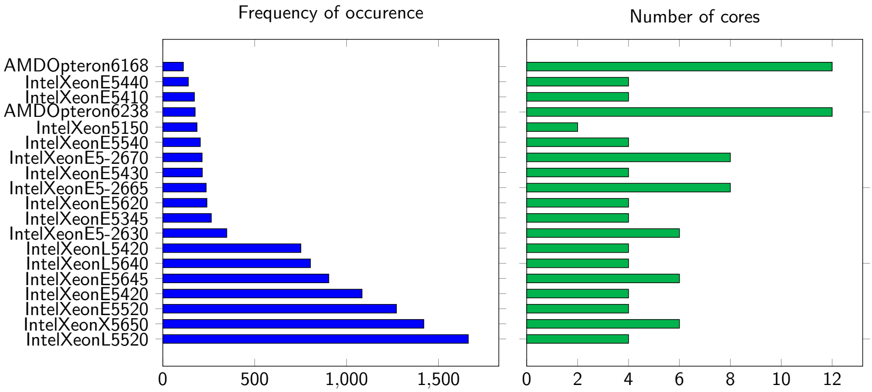


Figure: The 20 most common CPU types in the Worldwide LHC Computing Grid at the Tier-1 level (used by LHCb during reprocessing 2012)

Introduction

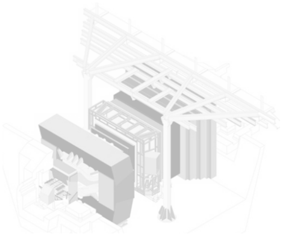
Main Problem: **Memory Footprint**

- Memory has constantly increased
- Throughput sometimes limited by memory

Two trends:

- Memory per core on future manycore system
- Increasing LHC beam energy

$\Longrightarrow$ Parallelization: Sharing of datasets



Introduction

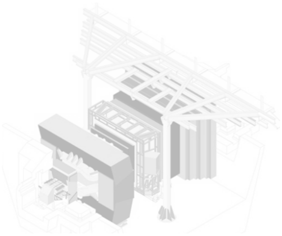
Main Problem: **Memory Footprint**

- Memory has constantly increased
- Throughput sometimes limited by memory

Two trends:

- Memory per core on future manycore system
- Increasing LHC beam energy

$\Longrightarrow$ Parallelization: Sharing of datasets



Introduction

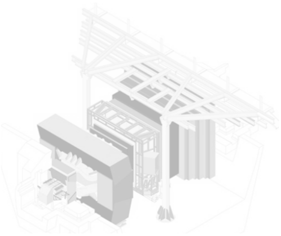
Main Problem: **Memory Footprint**

- Memory has constantly increased
- Throughput sometimes limited by memory

Two trends:

- Memory per core on future manycore system
- Increasing LHC beam energy

⟹ Parallelization: Sharing of datasets



Introduction

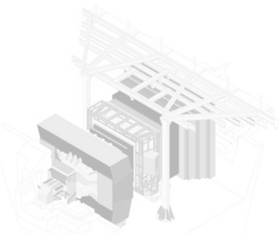
Main Problem: **Memory Footprint**

- Memory has constantly increased
- Throughput sometimes limited by memory

Two trends:

- Memory per core on future manycore system
- Increasing LHC beam energy

$\Longrightarrow$ Parallelization: Sharing of datasets



Introduction

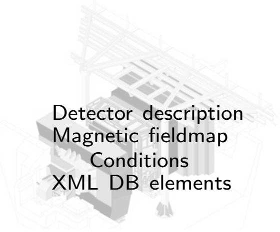
Main Problem: **Memory Footprint**

- Memory has constantly increased
- Throughput sometimes limited by memory

Two trends:

- Memory per core on future manycore system
- Increasing LHC beam energy

$\Longrightarrow$ Parallelization: Sharing of datasets



Introduction

First step: **Parallelization of software**

- GaudiMP
- AthenaMP
- GaudiHive
- multithreaded CMSSW
- Geant4
- ...

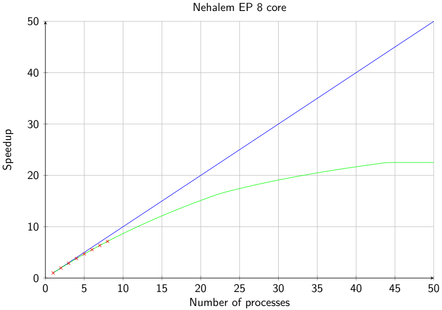
Introduction

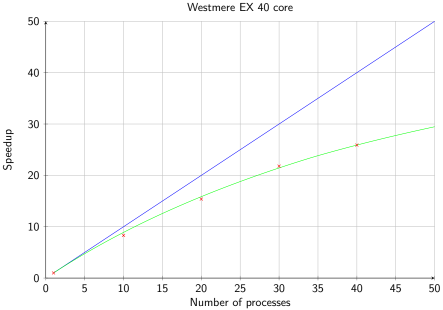
First step: **Parallelization of software**

- GaudiMP
- AthenaMP
- GaudiHive
- multithreaded CMSSW
- Geant4
- ...

Introduction

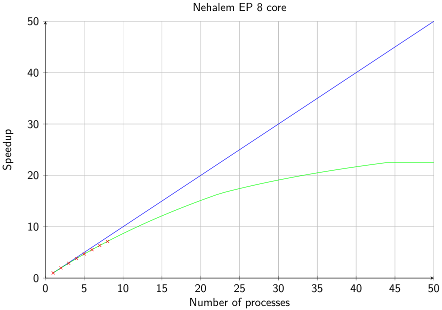
Speedup of LHCb parallel reconstruction jobs:

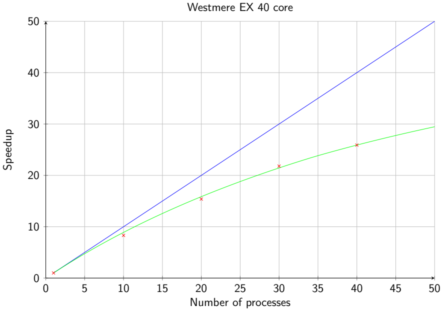




We probably don't want to assign all cores to such a job

Speedup of LHCb parallel reconstruction jobs:





We probably don't want to assign all cores to such a job

Problems

- Limit the number of processes for each job
- Jobs scale differently on different micro architectures
- Job options and characteristics of events impact runtime and speedup
- Grid site or experiment problem?

What we need:

- Scheduler within experiment's WMS, which takes care of:
  - Runtime prediction
  - Job properties (number of processes)
  - Optimize scheduling decision such that overall throughput increases
  - Backfilling

Problems

- Limit the number of processes for each job
- Jobs scale differently on different micro architectures
- Job options and characteristics of events impact runtime and speedup
- Grid site or experiment problem?

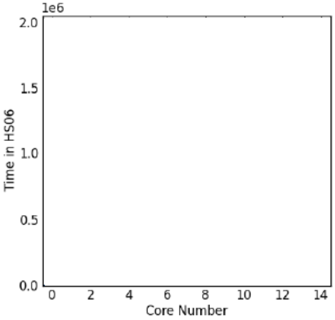
What we need:

- Scheduler within experiment's WMS, which takes care of:
  - Runtime prediction
  - Job properties (number of processes)
  - Optimize scheduling decision such that overall throughput increases
  - Backfilling

Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria

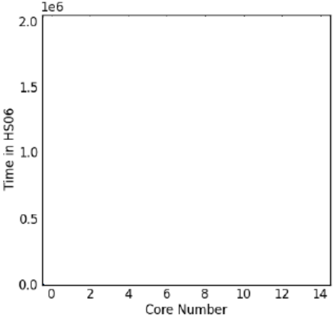
Objective Function:



Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria

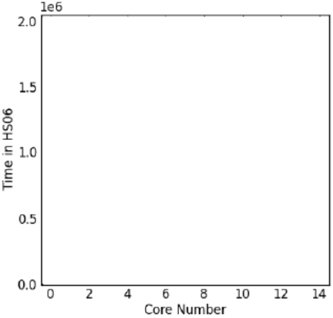
Objective Function:



Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria

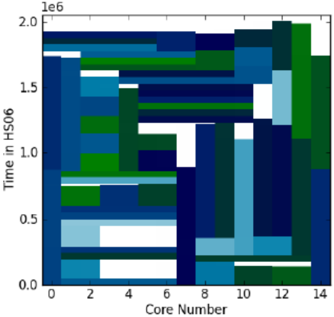
Objective Function: C



Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria.

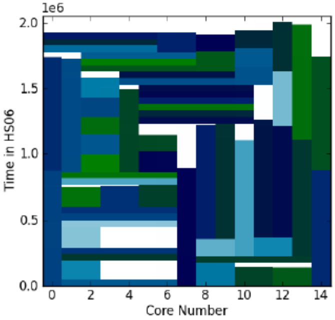
Objective Function: $$\underbrace{C - \sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j}_{Lost\ due\ to\ gaps}$$

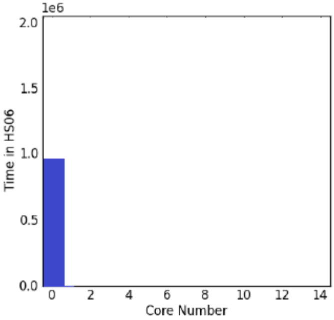


Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria

Objective Function: $\underbrace{C - \sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j}_{Lost\ due\ to\ gaps}$

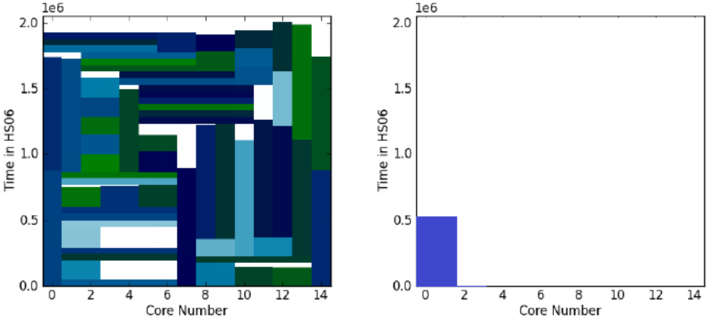




Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria

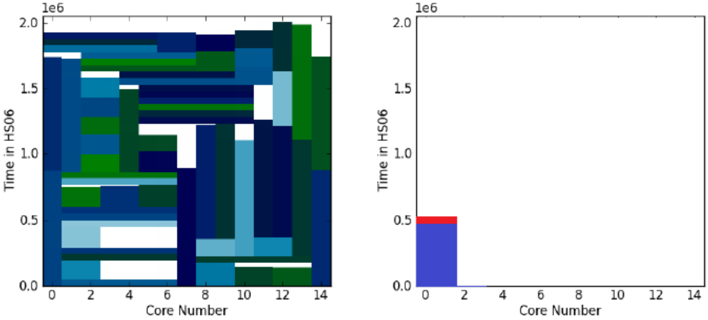
Objective Function: $\underbrace{C - \sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j}_{Lost\ due\ to\ gaps}$



Definition of the scheduling problem

Moldable job model: A scheduler has to choose the appropriate degree of parallelism for a job depending on certain criteria

Objective Function: $$\underbrace{C - \sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j}_{Lost\ due\ to\ gaps} + \underbrace{\sum_{j=1}^{J} \left(\frac{time_j}{s(n_j)} - \frac{time_j}{n_j} \right) \cdot n_j}_{Lost\ due\ to\ non\ linear\ speedup}$$



Optimizing the scheduling problem

1. Predict runtime, memory demand, speedup for each job
2. Define degree of parallelism of each job such that

$$\frac{MemoryFootprint}{p} < \frac{RAM}{NumberOfJobSlots}$$

3. Order list of jobs
4. Define schedule
5. Increase partition size of single jobs OR modify position within the schedule: if objective function improves keep the modification

In order to solve step 5: Constraint Programming, Local Search Methods, Probabilistic Methods

Optimizing the scheduling problem

1. Predict runtime, memory demand, speedup for each job
2. Define degree of parallelism of each job such that

$$\frac{MemoryFootprint}{p} < \frac{RAM}{NumberOfJobSlots}$$

3. Order list of jobs
4. Define schedule
5. Increase partition size of single jobs OR modify position within the schedule: if objective function improves keep the modification

In order to solve step 5: Constraint Programming, Local Search Methods, Probabilistic Methods

Optimizing the scheduling problem

1. Predict runtime, memory demand, speedup for each job
2. Define degree of parallelism of each job such that

$$\frac{MemoryFootprint}{p} < \frac{RAM}{NumberOfJobSlots}$$

3. Order list of jobs
4. Define schedule
5. Increase partition size of single jobs OR modify position within the schedule: if objective function improves keep the modification

In order to solve step 5: Constraint Programming, Local Search Methods, Probabilistic Methods

Optimizing the scheduling problem

1. Predict runtime, memory demand, speedup for each job
2. Define degree of parallelism of each job such that

$$\frac{MemoryFootprint}{p} < \frac{RAM}{NumberOfJobSlots}$$

3. Order list of jobs
4. Define schedule
5. Increase partition size of single jobs OR modify position within the schedule: if objective function improves keep the modification

In order to solve step 5: Constraint Programming, Local Search Methods, Probabilistic Methods

Optimizing the scheduling problem

1. Predict runtime, memory demand, speedup for each job
2. Define degree of parallelism of each job such that

$$\frac{MemoryFootprint}{p} < \frac{RAM}{NumberOfJobSlots}$$

3. Order list of jobs
4. Define schedule
5. Increase partition size of single jobs OR modify position within the schedule: if objective function improves keep the modification

In order to solve step 5: Constraint Programming, Local Search Methods, Probabilistic Methods

Optimizing the scheduling problem

1. Predict runtime, memory demand, speedup for each job
2. Define degree of parallelism of each job such that

$$\frac{MemoryFootprint}{p} < \frac{RAM}{NumberOfJobSlots}$$

3. Order list of jobs
4. Define schedule
5. Increase partition size of single jobs OR modify position within the schedule: if objective function improves keep the modification

In order to solve step 5: Constraint Programming, Local Search Methods, Probabilistic Methods

Constraint Programming: IBM Cplex Solver

Build a tree with all possible combinations

- Each leaf = 1 solution = 1 schedule

- Constraint propagation:

$$\sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j \leq C \qquad \left(\sum_{j=1}^{J} n_j \cdot job_j.running(t) \right) \leq nCores \qquad \forall t \ in \ [0, t_{max}]$$

Constraint Programming: IBM Cplex Solver

Build a tree with all possible combinations

- Each leaf = 1 solution = 1 schedule
- Constraint propagation:

$$\sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j \leq C \qquad \left(\sum_{j=1}^{J} n_j \cdot job_j.running(t) \right) \leq nCores \qquad \forall t \ in \ [0, t_{max}]$$

Constraint Programming: IBM Cplex Solver

Build a tree with all possible combinations

- Each leaf = 1 solution = 1 schedule
- Constraint propagation:

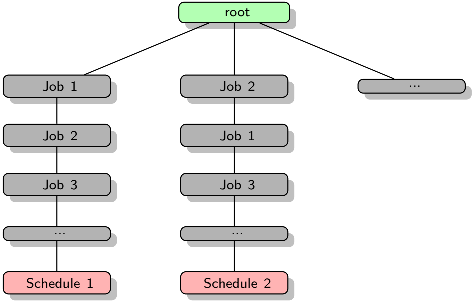
$$\sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j \leq C \qquad \left(\sum_{j=1}^{J} n_j \cdot job_j.running(t) \right) \leq nCores \qquad \forall t \ in \ [0, t_{max}]$$

Constraint Programming: IBM Cplex Solver

Build a tree with all possible combinations

- Each leaf = 1 solution = 1 schedule
- Constraint propagation:

$$\sum_{j=1}^{J} \frac{time_j}{s(n_j)} \cdot n_j \leq C \qquad \left(\sum_{j=1}^{J} n_j \cdot job_j.running(t) \right) \leq nCores \qquad \forall t \ in \ [0, t_{max}]$$



Local Search: Hill Climbing

Define a start schedule

1. Create a list of *candidates*
2. Pick the next *candidate*, increase its number of processes by $+1$
3. Define new schedule:
    - if throughput increases keep solution
    - if not remove item from *candidates*
4. Repeat step 2-3 until no *candidates* left

Local Search: Hill Climbing - Test run

video.mp4

video.mp4

(o) Start Schedule
Loss in throughput: 14.2%
Placed jobs: 126
Link 1

(p) Optimize Schedule
Loss in throughput: 3.1%
Placed jobs: 128
Link 2

Probabilistic Local Search: Simulated Annealing

Similar to Hill Climbing, but:

- Create more random solutions
- Accept worse solutions with certain probability
- Acceptance probability decreases over time

Comparison

	Constraint Programming	Hill Climbing	Simulated Annealing
Solution	Global Optima	Local Optima	Local Optima
Memory	runs easily out of memory	hundreds MB	hundreds MB
Runtime	several days	few minutes	depends on parameters

Mix of both to find better local optima

Comparison

	Constraint Programming	**Hill Climbing**	**Simulated Annealing**
Solution	Global Optima	Local Optima	Local Optima
Memory	runs easily out of memory	hundreds MB	hundreds MB
Runtime	several days	few minutes	depends on parameters

Mix of both to find better local optima

Prediction of runtime, memory, speedup

Problems:

- Estimation of job requirements is important
- Production manager does it by hand
- Underestimation: jobs will be killed
- Overestimation: what to do with the remaining time

Solution:

- A lot of data from prior jobs
- Find correlations
- Define a history based estimation

Prediction of runtime, memory, speedup

Problems:

- Estimation of job requirements is important
- Production manager does it by hand
- Underestimation: jobs will be killed
- Overestimation: what to do with the remaining time

Solution:

- A lot of data from prior jobs
- Find correlations
- Define a history based estimation

Prediction of runtime, memory, speedup

Most important features - Runtime:

- Average multiplicity
- Size of input file
- Number of events
- Average event size
- Normalization factor of worker node

Most important features - Memory:

- File size
- Number of events
- But: cannot draw many conclusions from data (virtual memory)

Speedup: Inferred

Prediction of runtime, memory, speedup

Most important features - Runtime:

- Average multiplicity
- Size of input file
- Number of events
- Average event size
- Normalization factor of worker node

Most important features - Memory:

- File size
- Number of events
- But: cannot draw many conclusions from data (virtual memory)

Speedup: Inferred

Prediction of runtime, memory, speedup

Most important features - Runtime:

- Average multiplicity
- Size of input file
- Number of events
- Average event size
- Normalization factor of worker node

Most important features - Memory:

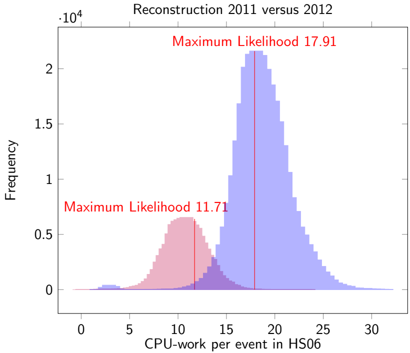
- File size
- Number of events
- But: cannot draw many conclusions from data (virtual memory)

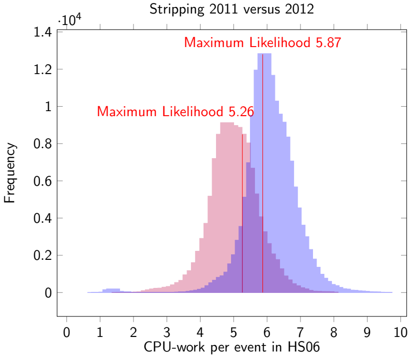
Speedup: Inferred

Run time prediction

Analysing LHCb's reprocessing productions from 2011 versus 2012:

$$CPUTime \cdot HEPSPECValue / NumberOfEvents$$





Runtime prediction

With linear regression runtime prediction can be improved up to 20% compared to MLE

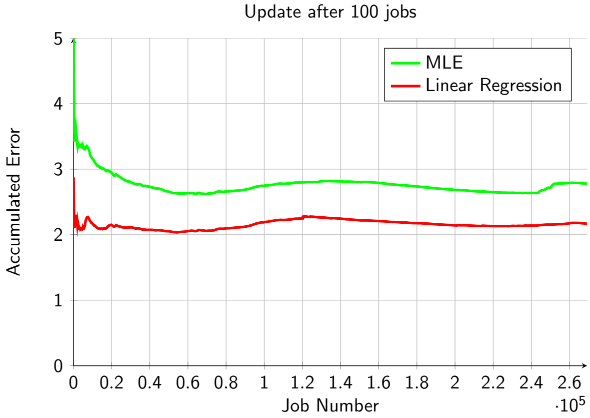
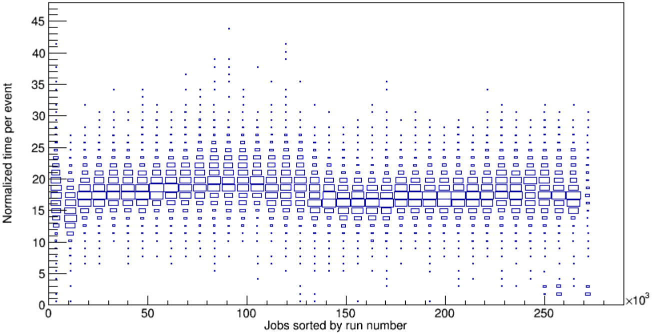


Figure: Accumulated error for the prediction of runtime

Runtime prediction

Distribution of runtime values per event sorted by run number:



Questions?