Evaluating moldability of LHCb jobs for multicore job submission

Nathalie Rauschmayr

On behalf of LHCb collaboration

September 26, 2013

- 1 Introduction
- 2 Scheduling and multicore jobs
- 3 Example
 - Example - How to find an optimum?
- 4 Speedup prediction
- 5 Run time prediction
 - Stripping
 - Reconstruction
- 6 Back to the example
- 7 Outcome

- 1 Introduction
- 2 Scheduling and multicore jobs
- 3 Example
 - Example - How to find an optimum?
- 4 Speedup prediction
- 5 Run time prediction
 - Stripping
 - Reconstruction
- 6 Back to the example
- 7 Outcome

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Introduction

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Introduction

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Introduction

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Introduction

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Introduction

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Introduction

LHCb experiment:

- One of the large LHC experiments
- Difference in Matter and Antimatter (CP Violation)
- B Physics

Main Problem: **Memory Footprint**

Two trends:

- Memory per core
 - Currently: 0.5 bytes/flop
 - Foreseen: < 0.1 bytes/flop
- LHC parameters
 - Larger events (30 kB in 2009 up to 60 kB in 2012)
 - Complexity of reconstruction

Sharing of datasets:

Sharing of datasets:

The more processes the larger the overall memory reduction:

$$(n - 1) \cdot \textit{SharedMemory}$$

Sharing of datasets:

The more processes the larger the overall memory reduction:

$$(n - 1) \cdot \textit{SharedMemory}$$

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 8 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 8 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 8 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 8 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 8 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 50 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling
 - Mix jobs in an appropriate way

- Scheduling becomes more complex
- One more dimension: Number of processes
- Example: Worker node with 50 cores available for 1 day
- What to do?
 - Run each job with all cores available → reasonable if good scaling
 - Mix jobs in an appropriate way

Scheduling and multicore jobs

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: Run time prediction

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: Run time prediction

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: Run time prediction

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

Scheduling and multicore jobs

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: **Run time prediction**

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: **Run time prediction**

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: Run time prediction

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

Scheduling and multicore jobs

How to use multicore CPUs more efficiently:

- Reduce gaps in schedule
- Limit loss due to non linear speedup

→ Use moldability of jobs to optimize objective function

Main Problem: Run time prediction

- What does run time rely on?
- Can it be predicted within a given range?
- How large is the introduced error?

Example

Input: Certain number of jobs and cores

Test: All possible combinations

Example

Input: Certain number of jobs and cores

Test: All possible combinations

Figure : Used total CPU-time with different mixtures of parallel jobs

Example

Input: Certain number of jobs and cores

Test: All possible combinations

Figure : Used total CPU-time with different mixtures of parallel jobs

Can the optimum be predicted?

How is the result influenced by errors in run time prediction?

Example - How to find an optimum?

- NP hard problem
- Iterative approaches:
 - Assign an additional core to the most worthy job
 - Job which loses less CPU time due to non linear speedup
- Required input:
 - Speedup
 - Run time prediction

Speedup prediction

Downey Speedup Model:

$$S(n) = \begin{cases} \frac{An}{A+\sigma(n-1)/2} & 1 \leq n \leq A \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)} & A \leq n \leq 2A-1 \\ A & n \geq 2A-1 \end{cases}$$

Speedup prediction

Downey Speedup Model:

$$S(n) = \begin{cases} \frac{An}{A+\sigma(n-1)/2} & 1 \leq n \leq A \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)} & A \leq n \leq 2A-1 \\ A & n \geq 2A-1 \end{cases}$$

Figure : A = Average parallelism and σ = variance in parallelism

Run time prediction

- Using historical information
- Clustering
 - Grid Site
 - CPU-type
 - Eventtype
 - Production
 - Workernode
- Distribution of datasets: CPU-work per event (CPU-time · HEPSPec-value)

$$RunTime = nEvt \cdot MaxLikelihood / PowerOfMachine$$

Run time prediction - Stripping

Figure : Stripping jobs of reprocessing 2011 versus 2012

Run time prediction - Reconstruction

Figure : Reconstruction jobs of reprocessing 2011 versus 2012

Run time prediction - Simulation step

Figure : Maximum likelihoods of MC11 jobs with different event types and from different productions

Back to the example

Run time predicted as:

$$Workload_{min} = MaxLikelihood - x \cdot \sigma$$

$$Workload_{max} = MaxLikelihood + x \cdot \sigma$$

Back to the example

Run time predicted as:

$$Workload_{min} = MaxLikelihood - x \cdot \sigma$$

$$Workload_{max} = MaxLikelihood + x \cdot \sigma$$

(a) Confidence interval of 0σ

(b) Confidence interval of 2σ

Figure : Decision found by an iterative approach

- Difficult to predict the optimum
- Theoretical optimum must not be the real one:
 - Jobs can influence each other (concurrent accesses)
 - Uncertainties in the prediction (LHC configuration)
 - Iterative approaches tend get stuck in local optima
- Approximation of global optimum already sufficient

- Difficult to predict the optimum
- Theoretical optimum must not be the real one:
 - Jobs can influence each other (concurrent accesses)
 - Uncertainties in the prediction (LHC configuration)
 - Iterative approaches tend get stuck in local optima
- Approximation of global optimum already sufficient

- Difficult to predict the optimum
- Theoretical optimum must not be the real one:
 - Jobs can influence each other (concurrent accesses)
 - Uncertainties in the prediction (LHC configuration)
 - Iterative approaches tend get stuck in local optima
- Approximation of global optimum already sufficient

- Difficult to predict the optimum
- Theoretical optimum must not be the real one:
 - Jobs can influence each other (concurrent accesses)
 - Uncertainties in the prediction (LHC configuration)
 - Iterative approaches tend get stuck in local optima
- Approximation of global optimum already sufficient

- Difficult to predict the optimum
- Theoretical optimum must not be the real one:
 - Jobs can influence each other (concurrent accesses)
 - Uncertainties in the prediction (LHC configuration)
 - Iterative approaches tend get stuck in local optima
- Approximation of global optimum already sufficient

- Difficult to predict the optimum
- Theoretical optimum must not be the real one:
 - Jobs can influence each other (concurrent accesses)
 - Uncertainties in the prediction (LHC configuration)
 - Iterative approaches tend get stuck in local optima
- Approximation of global optimum already sufficient