Computing the expected longest path of task graphs in the presence of silent errors

Henri Casanova[1], Julien Herrmann[2], **Yves Robert**[3,4]

1. University of Hawaii at Maona, Honululu, USA
2. Georgia Tech, Atlanta, USA
3. ENS Lyon, France
4. University of Tennessee Knoxville

ICL Friday Lunch Talk – September 9, 2016

Outline

Outline

Motivation

Why do we need to compute longest paths in graphs?

- HPC applications = computational workflows:
  $\hookrightarrow$ Vertices: tasks with execution time $\mathbf{x_i} \in \mathbb{R}^+$
  $\hookrightarrow$ Edges: data dependencies

- List scheduling:
  $\hookrightarrow$ Critical Path Scheduling (based on bottom-levels)
  $\hookrightarrow$ HEFT algorithm (for heterogeneous environments)

Motivation

- Execute workflows at large scale $\Rightarrow$ resilience!
- Decide which task to checkpoint?
- Intuition: checkpoint more frequently on critical paths
- Need to compute expected path lengths ☹ ☹ ☹

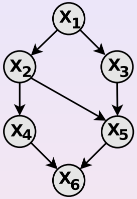
Known results

- Linear chains: dynamic programming algorithms ☺
- Linearizing general DAGs:
    - Execute tasks in sequence (on whole platform)
    - Decide for both ordering and checkpoints
    - Intuition: complicated!
      need to store a wavefront of checkpointed data
    - Already NP-hard for fork graphs ☹
- Nothing known to execute DAGs with concurrency and limited resources

Paths with probabilistic weights

Longest path:

$$L = X_6 + max(X_5 + max(X_2, X_3), X_4 + X_2) + X_1$$



- Deterministic weights:
  $\hookrightarrow$ Depth-first traversal: $O(|V| + |E|)$
- Random weights (PERT networks):
  $\hookrightarrow$ L is a random variable
  $\hookrightarrow$ Computing L's distribution: #P-complete
  $\hookrightarrow$ Computing L's expected value: #P-complete

Outline

Monte-Carlo approach

- For each task: weight is **sampled** from its probability distribution:

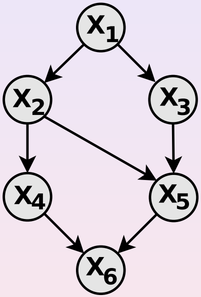
$$x_i \leftarrow X_i$$

- Longest path is computed:

$$L(x_1, x_2, ..., x_n)$$

- Repeat a large number of iterations

$\hookrightarrow$ gives an **empirical expected value**


$$\begin{tikzcd} & X_1 \arrow[dl] \arrow[dr] & \\ X_2 \arrow[d] \arrow[drr] & & X_3 \arrow[d] \\ X_4 \arrow[dr] & & X_5 \arrow[dl] \\ & X_6 & \end{tikzcd}$$

Approximation by a series-parallel graph

We deal with **independent** variables!

- $X_1 + X_2$: **convolution of density functions**

$$f_{X_1+X_2}(x) = \int_t f_{X_1}(t) f_{X_2}(x-t)dt$$

- $max(X_1, X_2)$: **product of cumulative functions**

$$F_{X_1 \times X_2} = F_{X_1} \times F_{X_2}$$

$$f_{X_1 \times X_2}(x) = F_{X_1}(x) \times f_{X_2}(x) + f_{X_1}(x) \times F_{X_2}(x)$$

Exact results on series-parallel graphs.

Dodin algorithm on general graphs [Op. Research, 1985]

Approximation by a series-parallel graph.

Approximation with normality assumption

Clark's formula on two **dependent** normal laws:

$$X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2) \qquad X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

Approximation of the Sum and Max by a normal law.

- Sum of corollated normal laws: $X_3 = X_1 + X_2$
  - Expected value: $\mu_3 = \mu_1 + \mu_2$
  - Variance: $\sigma_3^2 = \sigma_1^2 + 2.\sigma_1^2.\sigma_2^2.\rho_{X_1,X_2} + \sigma_2^2$
  - Correlation coefficient: closed-form formula
- Max of corollated normal laws: $X_3 = X_1 \times X_2$
  - (Complicated) closed-form formulas

Normal approximation on general graphs [Op. Research, 1983]

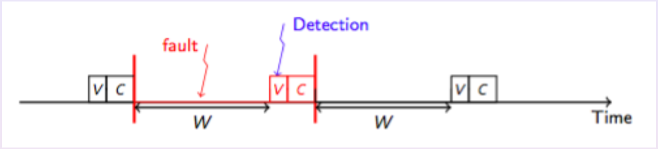
Consider that every random variable is normally distributed.

Outline

Silent errors

- Silent Data Corruptions
  $\hookrightarrow$ major challenges for Exascale
  $\hookrightarrow$ cosmic radiations
  $\hookrightarrow$ packaging pollution
  $\hookrightarrow$ Dynamic Voltage Frequency Scaling
- Verification at the end of task
  $\hookrightarrow$ checksums (linear algebra kernels)
- Errors are independent and exponentially distributed
  $\hookrightarrow$ Mean Time Between Failure: $1/\lambda$

Verification at the end of task



# Re-executions	Total time
0	a_i
1	$2a_i$
2	$3a_i$
...	...

First order approximation

- Probability that an error occurs during the first execution of task i:

$$1 - e^{-\lambda a_i} = \lambda a_i + O(\lambda^2)$$

- Probability that an error occurs during the first **and** the second execution of task i:

$$(1 - e^{-\lambda a_i})^2 = O(\lambda^2)$$

- λ is small $\Rightarrow$ **first order approximation**

Model with first order approximation

For every task i :

$$X_i = \begin{cases} a_i & \text{with probability} \quad 1 - \lambda a_i \\ 2.a_i & \text{with probability} \quad \lambda a_i \end{cases}$$

Model with first order approximation

For every task i :

$$X_i = \begin{cases} a_i & \text{with probability} \quad 1 - \lambda a_i \\ 2.a_i & \text{with probability} \quad \lambda a_i \end{cases}$$

Theorem

Computing the expected longest path of a **probabilistic 2-state DAG** is a #P-complete problem.

$\mathcal{E}(G)$: expected longest path of G

- $\mathcal{E}(G)$ is a **polynomial** in λ
- $\mathcal{E}(G) = \sum_{S\subset V} P(S) \times L(S)$

$P(S) =$ probability if all tasks in S fail
$L(S) =$ longest path if and no task in $V \setminus S$ fails

$$P(\emptyset) = \prod_{i\in V}(1 - \lambda a_i + O(\lambda^2)) = 1 - \sum_{i\in V} \lambda a_i + O(\lambda^2)\ ,$$

$$P(\{i\}) = (\lambda a_i + O(\lambda^2)) \times \prod_{j\in V\setminus\{i\}} (1 - \lambda a_j + O(\lambda^2))$$

$$= \lambda a_i + O(\lambda^2)\text{ , and}$$

$$P(S) = O(\lambda^2) \text{ if } |S| > 1$$

$\mathcal{E}(G)$: expected longest path of G

- $\mathcal{E}(G) = L(G) + \lambda \sum_{i \in V} a_i(L(G_i) - L(G)) + O(\lambda^2)$

  where $L(G)$: deterministic longest path in G

  $L(G_i)$: deterministic longest path in G when task i has weight $2.a_i$

- First order approximation:

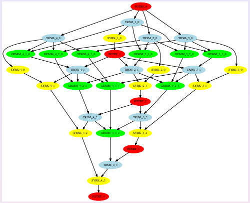
$$\mathcal{E}(G) = L(G) + \lambda \sum_{i \in V} a_i(L(G_i) - L(G))$$

$\hookrightarrow$ $(n+1)$ depth-first traversals: $O(|V|^2 + |V|.|E|)$

Outline

Experiments

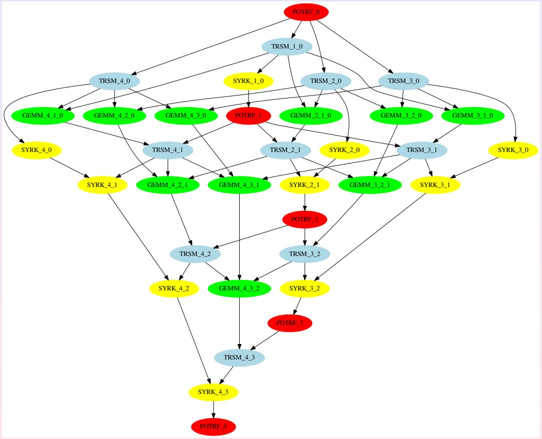
- Evaluation of:
  ↪ First order approximation
  ↪ Approximation by a series-parallel graph
  ↪ Approximation with normality assumption



- Comparison with Monte-Carlo approach
  ↪ 300,000 iterations

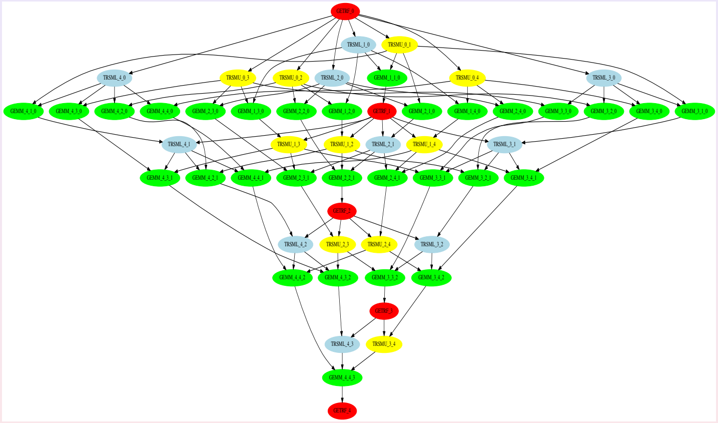
- DAGs from tiled Cholesky, LU and QR factorizations
  ↪ kernel execution times from StarPU

Application DAGs



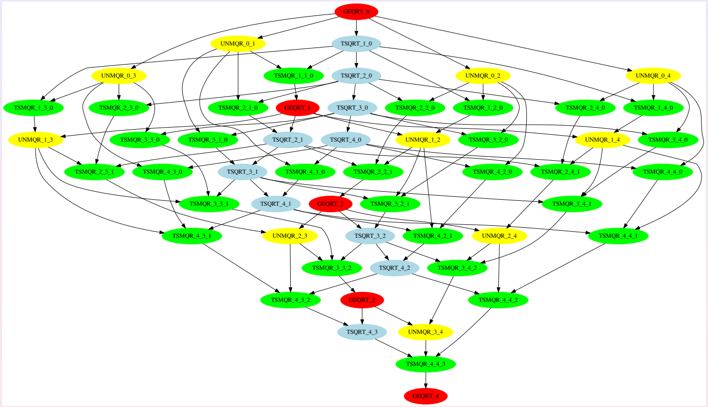
Cholesky factorization on a 5×5 tiled matrix

Application DAGs



LU factorization on a 5×5 tiled matrix

Application DAGs



QR factorization on a 5×5 tiled matrix

Application DAGs

- Tiled matrices of size $k \times k$
- k varied from 4 to $12 \Rightarrow$ up to 650 tasks
- Cholesky DAG has $\frac{1}{3}k^3 + O(k^2)$ tasks
- LU and QR DAGs have $\frac{2}{3}k^3 + O(k^2)$ tasks (twice more costly for QR)
- Weights = actual kernel execution times on Nvidia Tesla M2070 GPUs with tiles of size $b = 960$.

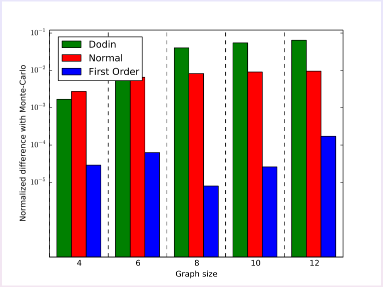
Error rates

- Consistent results across different DAGs
  $\Rightarrow$ Fix probability p_{fail} that an average-weight task will fail
- p_{fail} chosen as 0.01, 0.001, and 0.0001
- Technically: given $G = (V, E)$ and p_{fail}:
  - average task weight: $\bar{a} = \sum_{i \in V} a_i / |V|$
  - pick failure rate λ s.t. $p_{\text{fail}} = 1 - e^{-\lambda \bar{a}}$

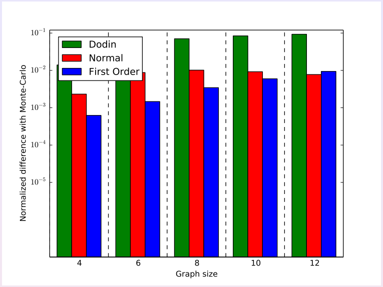
- Quite pessimistic p_{fail} values!
- For a platform with $100,000$ processors, processor MTBF is 17.27 days, 174 days and 4.7 years with these values
- Selected high p_{fail} values put our algorithm at a disadvantage with respect to its competitors

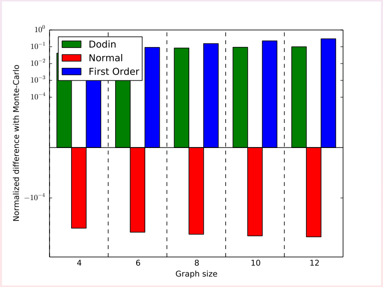
Results for QR factorization

$p_{\text{fail}} = 0.0001$



$p_{\text{fail}} = 0.001$

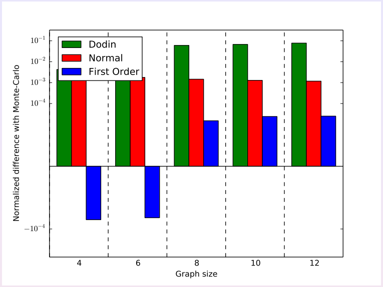




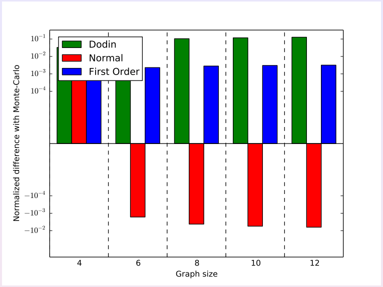
$p_{\text{fail}} = 0.01$

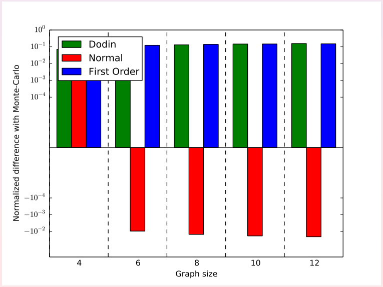
Results for LU factorization

$p_{\text{fail}} = 0.0001$



$p_{\text{fail}} = 0.001$





$p_{\text{fail}} = 0.01$

Scalability

	Dodin	Normal	First 0rder
Normalized difference with Monte Carlo	-0.97	954×10^{-6}	7×10^{-6}
Execution time	2 minutes	20 minutes	1 second

LU with $k = 20$ $(2,870$ tasks) and $p_{\text{fail}} = 0.0001$

Summary

- FirstOrder more accurate than both competitors ☺
- First Order much faster ☺ ☺ ☺
- Dodin approximation leads to higher error (DAGs far from series-parallel)

Outline

Conclusion

First order approximation

- First order approximation of expected longest path with silent errors
- Lower complexity than existing methods
- Better results for small (but realistic) failure rate

Perspectives

- Fail-stop errors
- Expected makespan for limited resources
- Still a long road before checkpointing DAGs ☹ ☹ ☹