Provenance Management in Many-Task Computing

Luiz Gadelha
CENAPAD-RJ – LNCC
lgadelha@lncc.br

First Brazil-France Workshop on High Performance Computing and Scientific Data Management Driven by Highly Demanding Applications

September 12, 2012

Laboratório
Nacional de
Computação
Científica

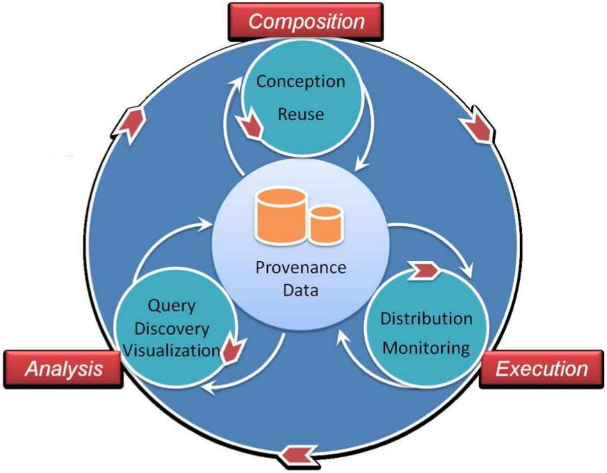
Agenda

Computational Scientific Experiments

- Scientific research increasingly rely on computational experiments.
- Properties of these experiments:
  - large number of computational tasks – *many-task computing* (MTC);
  - large amounts of data;
  - data stored in diverse formats;
  - use of parallel and distributed computing.

R. Kouzes et al. The Changing Paradigm of Data-Intensive Computing. *IEEE Computer*, 42(1):26-34, 2009.

Life Cycle of Computational Experiments



M. Mattoso et al. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management* 5(1):79–92, 2010.

Scientific Workflows

- A **scientific workflow** consists of the specification of a number of computational tasks along with their data dependencies.
- It follows the computational experiment life-cycle:
    - Composition, specification, data modeling.
    - Mapping and execution.
    - Provenance and metadata gathering.
- A **scientific workflow management system** (SWMS) allows for the management of workflow life-cycle.

E. Deelman et al. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems* 25(1):528-540, 2009.

Scientific Workflows

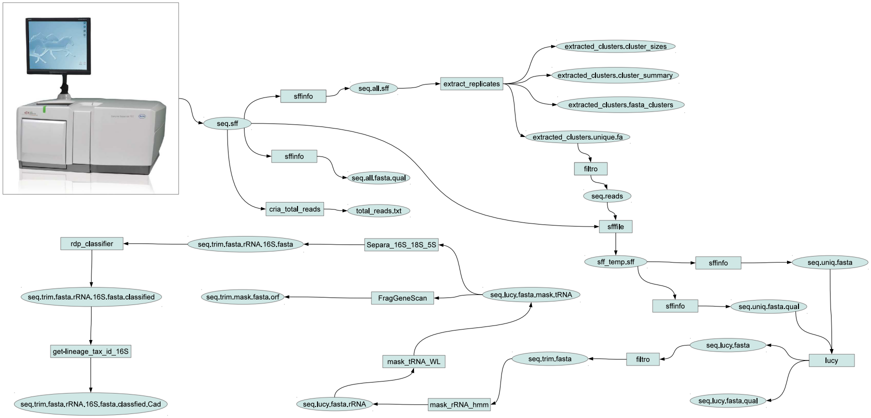
- Scientific workflows support the automation of computation scientific experiments:
  - task execution orchestration based on data dependencies;
  - data flow between tasks;
  - parallel execution of independent tasks;
  - task execution scheduling on HPC environemnts;
  - provenance management (gathering, storing, querying).

Example: Bioinformatics Laboratory, LNCC

Roche 454 FLX sequencer generates 12GB to 15GB per sequencing round.
Resulting data processed by various applications:

- quality filtering,
- data formatting,
- sequence alignment.

Example: Bioinformatics Laboratory, LNCC

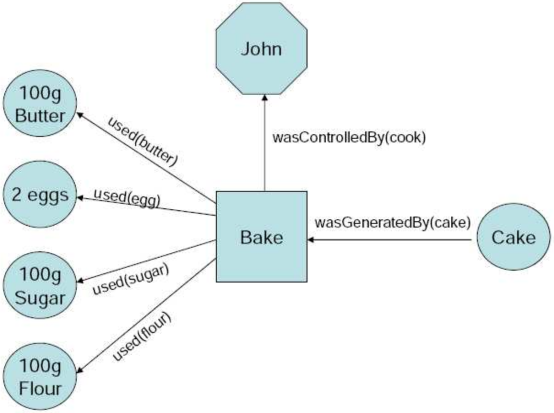


Provenance

- **Provenance** information describe the conception and execution history of a computational experiment.
- **Prospective** provenance captures specification and its evolution.
- **Retrospective** provenance describes dataset derivation history:
  - processes involved in the derivation;
  - other datasets used by these processes;
  - agents that controlled these processes.

S. B. Davidson and J. Freire, Provenance and scientific workflows: challenges and opportunities. *Proc. of the International Conference on Management of Data* (SIGMOD 2008), pp. 1345–1350. ACM, 2008.

Open Provenance Model



L. Moreau et al. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

Provenance Applications

- Applications of provenance:
  - intelligent re-execution of experiments;
  - authorship protection;

    L. Gadelha, M. Mattoso. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. *IEEE Fourth International Conference on e-Science* (e-Science 2008), pp. 597-602. 2008.
  - data quality evaluation;
  - reproducibility.

Large Scale Computational Scientific Experiments

- Execution should be scalable, i.e. resilient to increased number of tasks and amount of data.
- Usually require the use of parallel and distributed computing:
  - **Parallel computing**. Concurrent use of multiple processors to reduce execution time.

    I. Foster. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison-Wesley, 1994.
  - **Grid computing**. Sharing heterogeneous and distributed computational resources.

    I. Foster e C. Kesselman, Eds. The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 2003.
  - **Cloud Computing**. Computational resources delivered as a service.

    D. Oliveira, F. Baião e M. Mattoso. Towards a Taxonomy for Cloud Computing from an e-Science Perspective. Cloud Computing, pp. 47–62. Springer, 2010.

Scalability

- The total execution time of a computational task is usually given by:

$$t = t_{\text{computing}} + t_{\text{communicating}} + t_{\text{waiting}}$$

- There are many performance analysis tools for standalone applications.
- A challenge is to provide the same kind of tools for many-task computations.
- Scalability depends on adequate orchestration of computational tasks and efficient data management.

Analysis of Computational Scientific Experiments

- Provenance information can support the analysis phase.
- In the context of many-task computing, the mechanisms used for gathering and storing provenance can have a negative impact on scalability.
- However, the level of detail should be enough for proper analysis.

Analysis of Computational Scientific Experiments

- Provenance can support debugging and optimization of workflow execution and data management strategies, if enriched with adequate information.
- This is relevant in large-scale experiments, which are executed on HPC service providers where many users compete for processing time.
- Query interfaces for accessing this information are also important to enable experiment analysis.

Analysis of Computational Scientific Experiments

- Issues in managing provenance of many-task computations:
  - **Modeling**. Capturing experiment specification and execution history, taking into account issues important to scalability, such as resource consumption and fault tolerance.
  - **Usefulness**. Applications of provenance information, such as experiment analysis from the scientific standpoint and from the performance standpoint.
  - **Usability**. Allow for queries to be expressed in a relatively easy manner.

    H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. *Proc. SIGMOD 2007*, pp. 13–24.
  - **Security**. Determine adequate security controls for protecting provenance information.

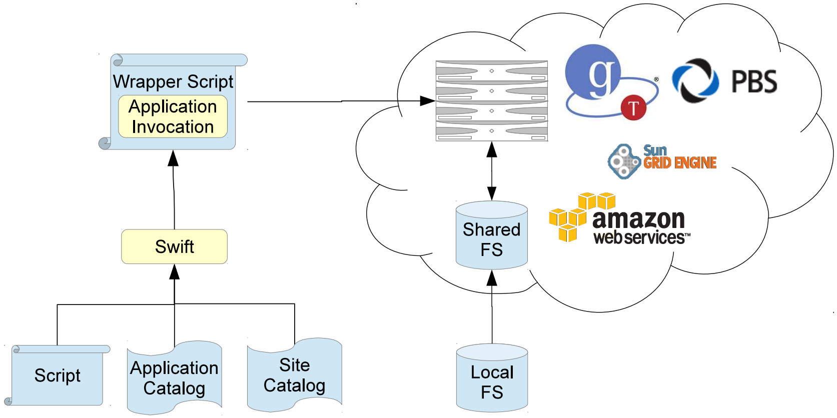
Swift

- Swift allows for the specification, execution, and analysis of many-task computations in parallel and distributed environments.
- Used in our experimentation since it scales to hundreds of thousands of CPUs.
- Components:
  - A high-level language for specifying scientific workflows as scripts.
  - Execution engine with native support for:
    - local execution,
    - parallel execution (e.g. PBS, SGE),
    - distributed execution (e.g. Condor, Globus).
  - Integrated provenance management system (MTCProv).

M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford I. Raicu, Parallel Scripting for Applications at the Petascale and Beyond. *IEEE Computer*, 42(11):50-60, 2009.

M. Wilde, M. Hategan, J. Wozniak, B. Clifford, D. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):634-652, 2011.

Swift: Execution Environment



Swift

- Development:

- Collaboration started during D.Sc. at COPPE/UFRJ with a graduate internship at Ian Foster's research group at the University of Chicago (March 2010 - February 2011).
- Scientific applications: OOPS (UChicago), SciColSim (UChicago), MODIS (UChicago), Bioinformatics Laboratory (LNCC).

MTCProv: Motivation

Provenance Challenge

- During the *Third Provenance Challenge* we presented a retrospective provenance model for MTC and its respective implementation in Swift.
- Problems detected:
  - Lack of domain specific information turned provenance information interpretation difficult.
  - Hard to write queries with generic query languages (SQL, XPath/XQuery, SPARQL).

L. Gadelha, B. Clifford, M. Mattoso, M. Wilde, and I. Foster. Provenance Management in Swift. *Future Generation Computer Systems*, 27(6):775-780, 2011.

MTCProv: Motivation

- MTCProv is a provenance management tool for parallel and distributed scientific workflows integrated to Swift.
- Development methodology:
  1. Survey of provenance query patterns.
  2. Data modeling.
  3. Implementation of gathering/storage mechanisms.
  4. Query interface implementation.

MTCProv: Query Patterns

- Patterns:
  - Entity Attribute (EA).
  - Direct Relationship (R).
  - Transitive Relationship (R^*).
  - Lineage Graph Matching (LGM).
  - Execution Summary (RS).
    - Performance (RRP).
    - Scientific (RSP).
  - Run Comparison (RCp).
  - Run Correlation (RCr).

L. Gadelha, M. Mattoso, M. Wilde, I. Foster, Provenance Query Patterns for Many-Task Scientific Computations. *Proceedings of the 3rd USENIX Workshop on Theory and Applications of Provenance* (TaPP'11), 2011.

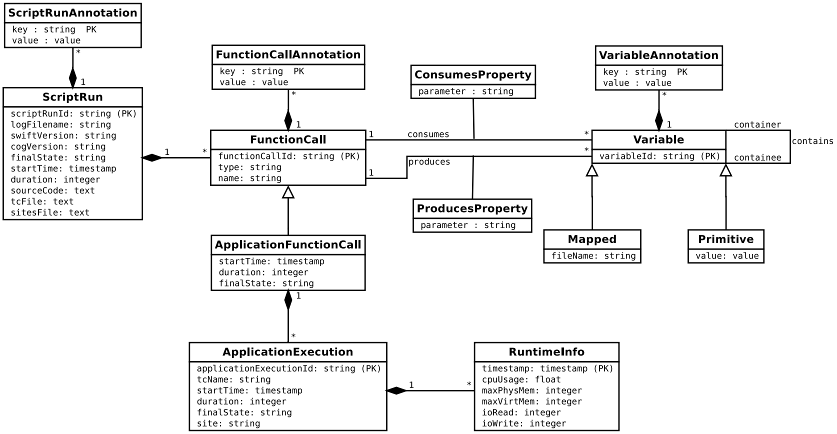
MTCProv: Patterns found in the *Provenance Challenges*

Pattern	PC1/PC2									PC3				PC3 (Optional Queries)														
	1	2	3	4	5	6	7	8	9	1	2	3	5	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EA	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
R	×	×	×		×	×	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×	×	×	×	×	×	×
R*	×	×	×			×	×			×	×	×	×		×		×				×	×			×	×	×	×
LGM							×								×													
RS	×	×	×							×	×	×	×	×		×	×						×	×	×	×	×	
RCp					×	×	×	×	×						×			×	×	×	×	×						×
RCr				×																								

MTCProv: Data Modeling

- Objectives:
  1. Gather consumption and production relationships between datasets and processes.
  2. Gather hierarchical relationships between datasets.
  3. Allow for the users to enrich their provenance records with annotations.
  4. Gather versioning information about scientific workflows and their component applications.
  5. Gather runtime information about external applications invoked from a Swift script.
  6. Provide a usable and useful query interface for provenance data.

MTCProv: Data Modeling

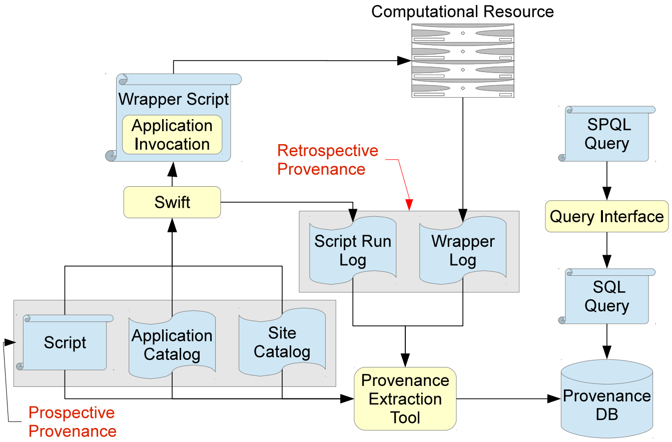


L. Gadelha, M. Wilde, M. Mattoso, and I. Foster. MTCProv: a practical provenance query framework for many-task scientific computing. *Distributed and Parallel Databases* 30(5-6):351-370. Springer, 2012.

MTCProv: Gathering and Storage

- Provenance information extracted on a per run basis from log files generated by Swift.
- Data stored in a relational database.
  - Annotations improve flexibility.
  - Transitive closure computed with recursive common table expressions defined SQL:1999.
- Graph-based data models require extensive traversals for computing aggregates.

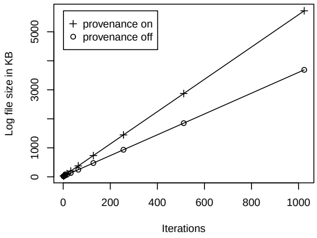
MTCProv: Gathering and Storage

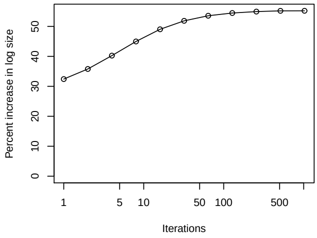


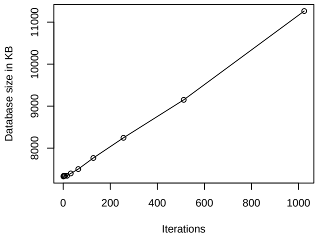
MTCProv: Gathering and Storage

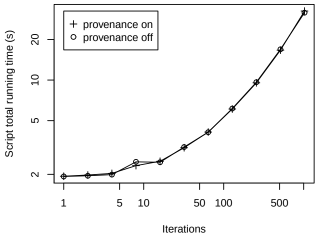
- Annotations can be generated by:
  - Ad-hoc scripts per application,
  - Wrapper scripts that trigger remote execution.
- Examples of annotations:
  - Scientific parameters usually opaque to Swift.
  - Digital signatures for protecting authorship.

MTCProv: Gathering and Storage Impact

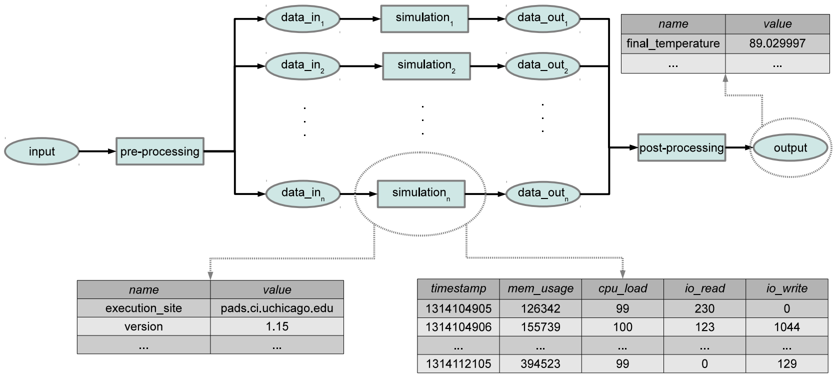








MTCProv: Annotation Gathering and Storage



name	*value*
final_temperature	89.029997
...	...

name	*value*
execution_site	pads.ci.uchicago.edu
version	1.15
...	...

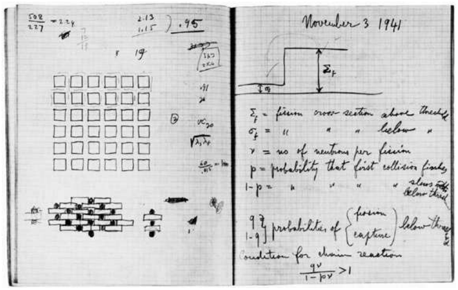
timestamp	*mem_usage*	*cpu_load*	*io_read*	*io_write*
1314104905	126342	99	230	0
1314104906	155739	100	123	1044
...	...	...	...	...
1314112105	394523	99	0	129

L. Gadelha, M. Wilde, M. Mattoso, and I. Foster. Exploring provenance in high performance scientific computing. Proc. Workshop on High Performance Computing Meets Databases (HPCDB'11), pp. 17–20, 2011.

MTCProv: Query Interface

- Abstraction of query patterns with functions and stored procedures in SQL.
- The R* is implemented with native recursion in SQL in the `ancestors` procedure.
- The RCp and RCr patterns are implemented by the function:
  - `compare_run`($\langle$ *list of parameters* or *annotation keys*$\rangle$) returns a table with values for parameters and annotations per script.
- A query interface was implemented in Java/ANTLR that automatically computes `FROM` clauses and join expressions.

MTCProv: Security



Σ_f
σ_f

November 3 1941

Σ_f = fission cross section above threshold
σ_f = " " " below "
ν = no of neutrons per fission
p = probability that first collision fissions
1-p = " " " " slows below threshold

$\left.\begin{matrix} q \\ 1-q \end{matrix}\right\}$ probabilities of $\left\{\begin{matrix} \text{fission} \\ \text{capture} \end{matrix}\right\}$ below threshold

Condition for chain reaction

$$\frac{q\nu}{1-p\nu} > 1$$

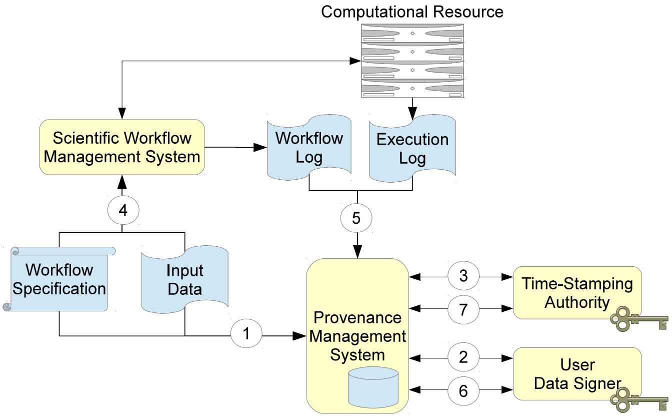
- Provenance records are analogous *laboratory notebooks*:
    - experiment plan;
    - initial parameters;
    - result description.

MTCProv: Security

- Recommendations for intellectual property protection in laboratory notebooks:
  - "The laboratory notebook is one of the most important elements in the patenting process."
  - "... one can see the importance of keeping a laboratory notebook to certify, and prove in a court if necessary, that your work that lead to an invention was performed before others."
  - "... the notes should be dated and signed by a third party."
  - "... after signature, no modification should be performed."

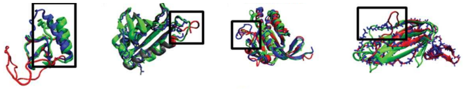
Guidelines for Maintaining a Lab Notebook, Los Alamos National Laboratory, 2012.

MTCProv: Security



L. Gadelha, M. Mattoso. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. *IEEE Fourth International Conference on e-Science* (e-Science 2008), pp. 597-602. 2008.

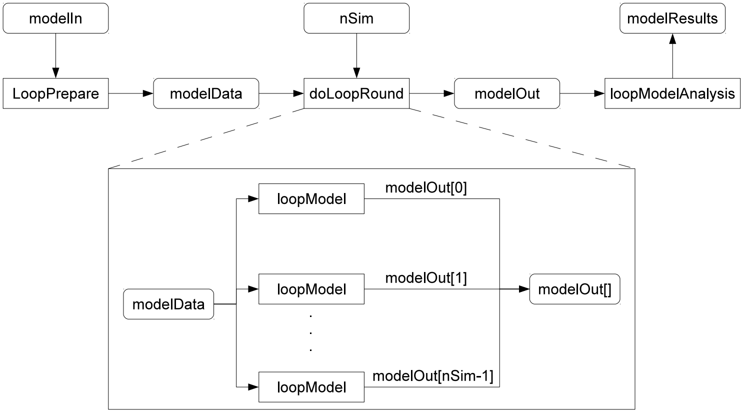
Case study: Open Protein Simulator



- Open Protein Simulator (OOPS) is a protein structure prediction application.
- Information visible to Swift:
    - Script-level scientific parameters (e.g. protein identifier).
    - Runtime execution statistics (e.g. duração de execuções).
- Ad-hoc scripts can collect:
    - File-level scientific parameters.
    - SVN versions of the script source code and application components.

A. Adhikari, J. Peng, M. Wilde, J. Xu, K. Freed, and T. Sosnick, Modeling large regions in proteins: Applications to loops, termini, and folding. *Protein Science* 21(1):107–121, 2012.

Case study: Open Protein Simulator



Case study: Open Protein Simulator

List executions between two dates:

```
select script_run
where  script_run.start_time between '2010-04-04' and '2010-08-08' and
       script_run.filename='psim.loops.swift';

                id                 |         start_time         |   ...
-----------------------------------+----------------------------+--------
 psim.loops-20100619-0339-b95ull7d | 2010-06-19 03:39:15.18-05  |   ...
 psim.loops-20100618-0402-qhm9ugg4 | 2010-06-18 04:02:21.234-05 |   ...
               ...                 |            ...             |   ...
```

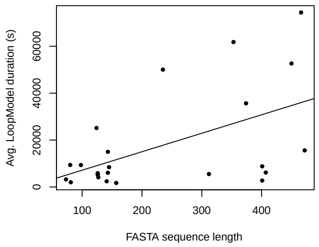
Case study: Open Protein Simulator

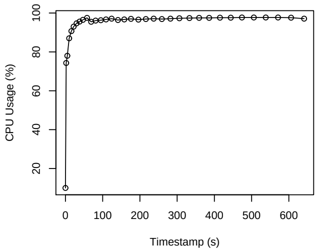
Correlate number of iterations and RMSD (scientific performance):

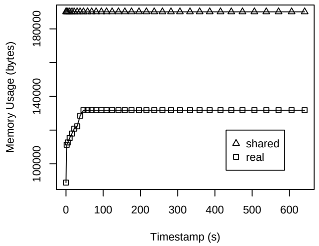
```
SELECT run_id, r.value as nSim, t.value as rmsd
FROM   compare_run_by_param('proteinId') as r
       INNER JOIN
       compare_run_by_param('nSim') as s USING (run_id)
       INNER JOIN
       compare_run_by_annot('rmsd') as t USING (run_id)
WHERE  r.value='TR567' and run.id LIKE 'psim.loops%';
```

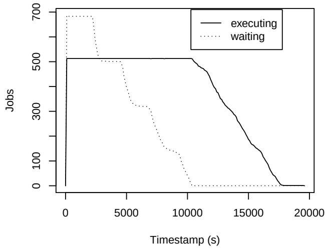
```
             run_id                | nSim |  rmsd
-----------------------------------+------+---------
 psim.loops-20100604-2215-cdifsnb3 |  256 | 3.33123
 psim.loops-20100613-0125-keyyyc35 |  512 | 0.76274
 psim.loops-20100616-1512-h6q4g4ja | 1024 | 0.68426
 ...
```

Case study: Open Protein Simulator



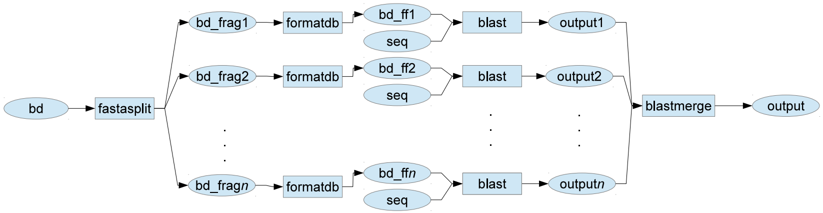






Case study: Parallel BLAST

Scientific worklow for parallelizing BLAST through input database partitioning:



Case study: Parallel BLAST

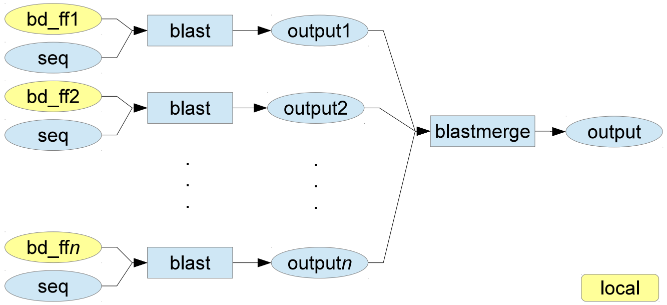
Analysis of waiting time in BLAST execution.

```
SELECT app_exec_id, timestamp, wait
FROM   runtime_info;

   app_exec_id         | timestamp  | wait
-----------------------+------------+-------
 blastall-3i191nsk     | 1339467548 | 100.0
 blastall-3i191nsk     | 1339467550 |  81.7
 blastall-3i191nsk     | 1339467552 |  50.8
 blastall-3i191nsk     | 1339467554 |  45.9
 blastall-3i191nsk     | 1339467555 |  38.4
 blastall-3i191nsk     | 1339467556 |  31.8
 blastall-3i191nsk     | 1339467558 |  30.0
 blastall-3i191nsk     | 1339467560 |  34.7
 blastall-3i191nsk     | 1339467561 |  33.1
```

$\Rightarrow$ I/O bottleneck.

Case study: Parallel BLAST



Case study: Parallel BLAST

Analysis of waiting time in BLAST execution.

```
SELECT app_exec_id, timestamp, wait
FROM   runtime_info;

   app_exec_id         | timestamp  | wait
----------------------+------------+-------
 blastall-leenrisk    | 1339924431 |   1.8
 blastall-leenrisk    | 1339924437 |   2.2
 blastall-leenrisk    | 1339924444 |   1.6
 blastall-leenrisk    | 1339924453 |   1.2
 blastall-leenrisk    | 1339924462 |   0.9
 blastall-leenrisk    | 1339924472 |   0.7
 blastall-leenrisk    | 1339924483 |   0.5
 blastall-leenrisk    | 1339924495 |   0.4
 blastall-leenrisk    | 1339924508 |   0.3
```

Related Work

- ParaTrac gathers performance data from workflow execution, however it does not gather information of the scientific domain. I/O data collected from FUSE and hierarchical process data from PROCFS.

  K. Dun et al. ParaTrac: a fine-grained profiler for data-intensive workflows. HPDC 2010, pp. 37–48.

- QLP can be used for provenance graph traversal however it does not express some query patterns easily.

  M. Anand et al. Approaches for Exploring and Querying Scientific Workflow Provenance Graphs. IPAW 2010, pp. 17–26.

- SPROV uses digital signatures for provenance record protection, however it does not deal with temporal information.

  R. Hasan et al. Preventing history forgery with secure provenance. ACM Transactions on Storage 5(4):1–43, 2009.

Concluding Remarks

- With MTCProv, we contributed with:
  - A provenance model that represents important information about parallelism and distribution of scientific workflows.
  - Gathering and storage of provenance with low impact on scientific workflow scalability.
  - Query interface that simplifies query expression through abstracting frequent patterns and simplifying joins.
  - Better support for computational experiment analysis allowing for result interpretation from the scientific and performance standpoints.
  - Identification of essential security controls for protecting provenance.

Concluding Remarks

- Future work:
  - Continue to explore applications of provenance to parallel and distributed workflow analysis.
  - Scalability of MTCProv as a provenance repository.
  - Survey of alternative query approaches, such as declarative languages (e.g. Datalog).
  - Exporing the use of different parallelization paradigms for component execution.
  - Integrate MTCProv to Swift's execution engine for online provenance use.

Conclusões

Merci! Thank you! Obrigado!