

An adaptive distributed query processing grid service

Fabio Porto EPFL-LBD - Switzerland

Vinícius F.V. da Silva

Márcio L. Dutra

Bruno Schulze

LNCC-Brazil

Outline

- Introduction
- Motivation
- General Architecture
- Query Optimization
- Adaptive Query Execution
- Initial Results
- Conclusion

Introduction

- Grid:
 - heterogeneous resources (hardware, software and data);
 - hard to predict node behavior;
 - middleware infra-structure grid services;
- Scientific applications:
 - large data volume;
 - user programs;
 - non-traditional data-types;
 - special operators and execution semantics;
- Grid services:
 - high-level services;
 - Integration with grid middleware;
 - adaptability to run-time conditions;
 - Web service communication model;

Motivating Problem

- Scientific Visualization Application (SVA)
 - Compute virtual particle trajectory flowing through a path (simplified model)
 - Inputs:
 - Domain geometry model;
 - Speed vectors;
 - Particles initial position;
 - Trajectory Computing program (TCP);

Problem Characteristics

- Can be modeled as a database problem:

Select part.part-id, **TCP**(part.position, **resulting-vector**(part.position,v.speed-vector))

From g in geometry, v in velocity,
part in particle

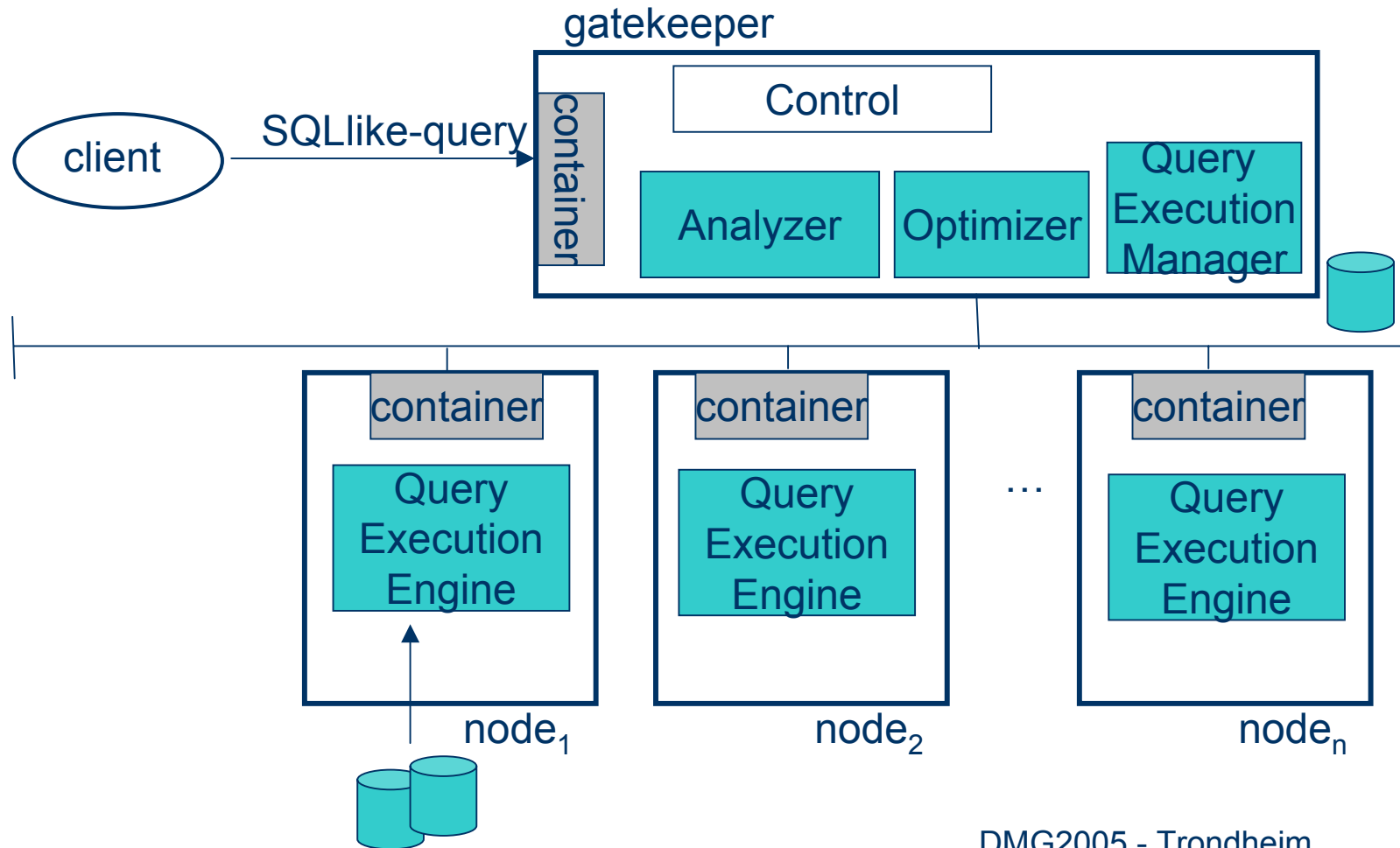
Where part.position in g.polyhedron and
g.idpolyhedron = v.idpolyhedron and
v.timeinstant \$time

- Large geometry (72000 tetrahedrons) and velocity datasets;
- Iteration over *time* values;
 - A path is computed out of a number of iterations for each particle;

Challenges

- Use grid available nodes to reduce query-elapsed-time for computing queries with user programs;
- Implement query processing components as grid services;
- Conceive a query optimization strategy that:
 - Parallelizes the execution of expensive fragments of a query execution plan (QEP);
 - Dynamically selects available heterogeneous grid nodes;
 - Supports iteration over a QEP fragment;
- Conceive a query execution strategy that:
 - Support parallelization of fragments of the QEP;
 - Adapts to fluctuations on nodes and network performance;
 - Adapts to variations on dataflow;
 - Supports iteration over a QEP fragment;
 - Support different nodes memory requirements;

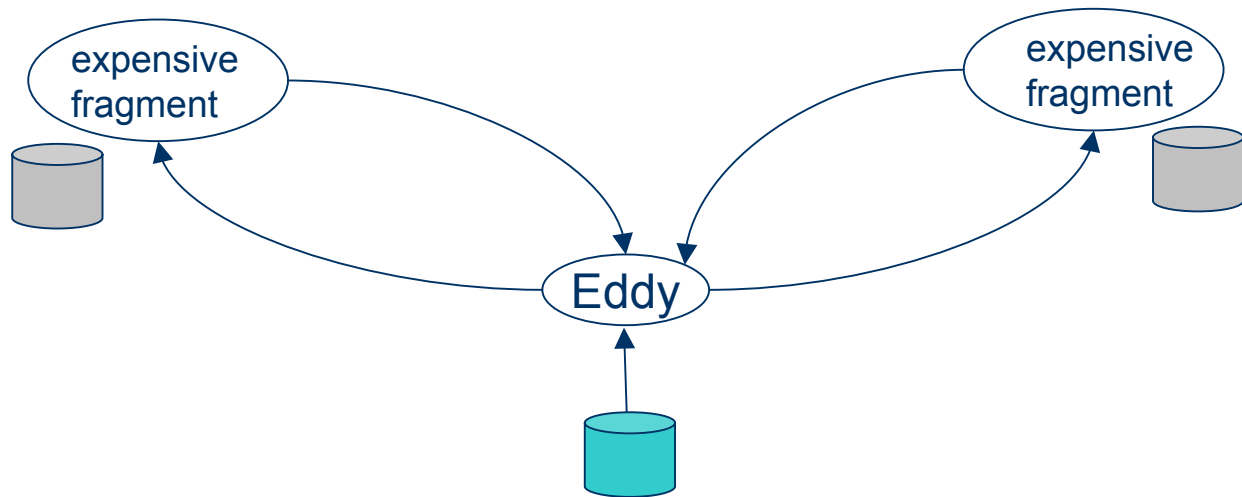
General Architecture



General Approach to Solution

- Adopt Eddy[Avnur00] strategy with the following objectives:
 - Adapt tuple routing through selected nodes;
 - Adapt size of a tuple block according to execution profile;
 - Implement tuple iteration over an expensive fragment of a QEP;

What do we want?



Query Optimization Strategy

- Query graph representation of user query
 - Dataflow constraints;
 - Usually with no more than 2 joins and few programs (≤ 3);
- Simple strategy for scientific queries:
 - Graph split into two sub-graphs with iteration border;
 - Order of joins decided through traditional dynamic programming within fragments;
 - Iteration border defines eddy placement in the QEP;
 - expensive fragment is parallelized;

Parallelizing expensive fragment

- Identifying interesting nodes for computing an expensive fragment of the QEP:
 - $N^i \subseteq N = \{n_1, n_2, \dots, n_m\}$, such that
 - $\text{Min}\{ \text{Max}_z \{c_1, c_2, \dots, c_j\} \}$
 - Greedy Grid node (G2N) scheduling algorithm;
 - Obtain historical node throughput info;
 - Order nodes by descending order of throughput;
 - Allocate all tuples to fastest node;
 - Allocate a new node if beneficial;
 - $\text{Max}\{c_1^* n_1, c_2^* n_2, \dots, c_{i-1}^* n_{i-1}\} > \text{Max}\{c_1^* n_1, c_2^* n_2, \dots, c_i^* n_i\}$

Beneficial node – cost model

- Cost model:
 - Initialization cost; Transfer cost; Fragment cost;
 - Initialization cost: $Tr * |G|$;
 - Transfer cost: $SC; Tr$ (block units); (bounded by SC)
 - Fragment cost: C_{ft} , $C_{bottleleneck}$; C_{lt} ;
 - Beneficial allocation considers:
 - Continuous evaluation of remote fragment
 - $|FC| * SB \geq 4 * SC$
 - $|B| > 4 * SC / |FC|$

Cost of a new node

- $|\# \text{ tuples}| < |BS|$
 - IT + sequential computation
- $|BS| < |\# \text{ tuples}| \leq 2 * |BS|$
 - IT + partial parallelization
- $|\# \text{ tuples}| > 2 * |BS|$
 - IT+full parallelization

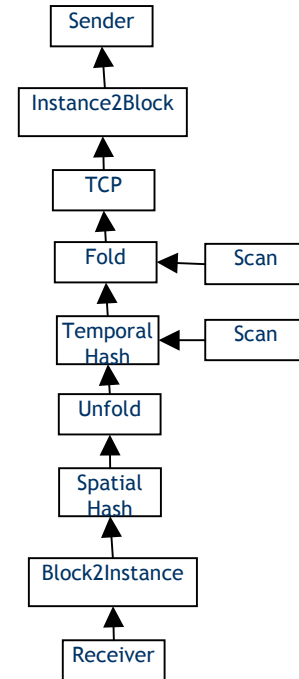
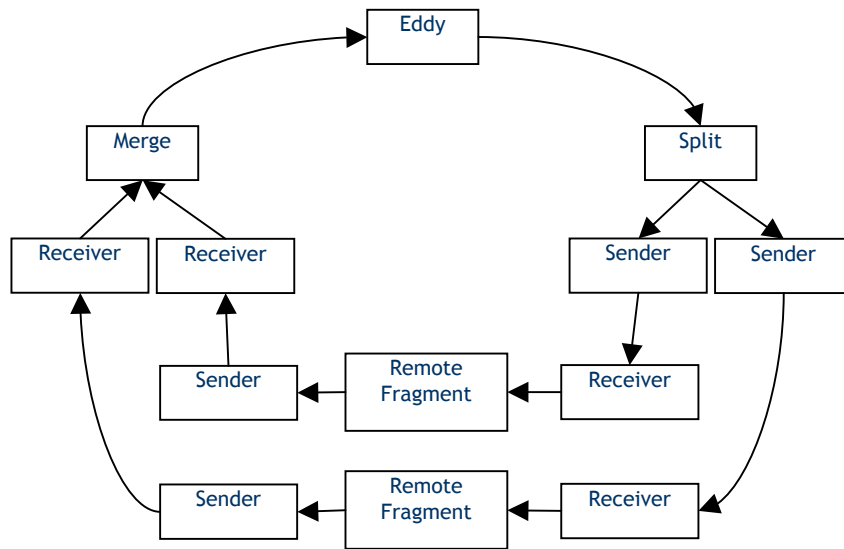
Query execution

- Extended QEEF[SBBD2003] query engine to cope with distribution and special operators semantics;
- Added the following physical operators:
 - Eddy,
 - Receiver/Sender,
 - Merge/Split,
 - Fold/unfold,
 - Block2Instance/Instance2Block,
 - Spatial hash-join,
 - Apply

Adapting Execution

- Small adaptation
 - Sender/Receiver in Eddy share data about tuple block cycle time;
 - Adapts block size accordingly;
 - Timeout in split, reduce block size;
 - 1/3 of data left, reduces to 2 allocated nodes;
 - Order for answering to getNext() requests;
- Large adaptation
 - Overall performance below a threshold – G2N

Query Execution Plans

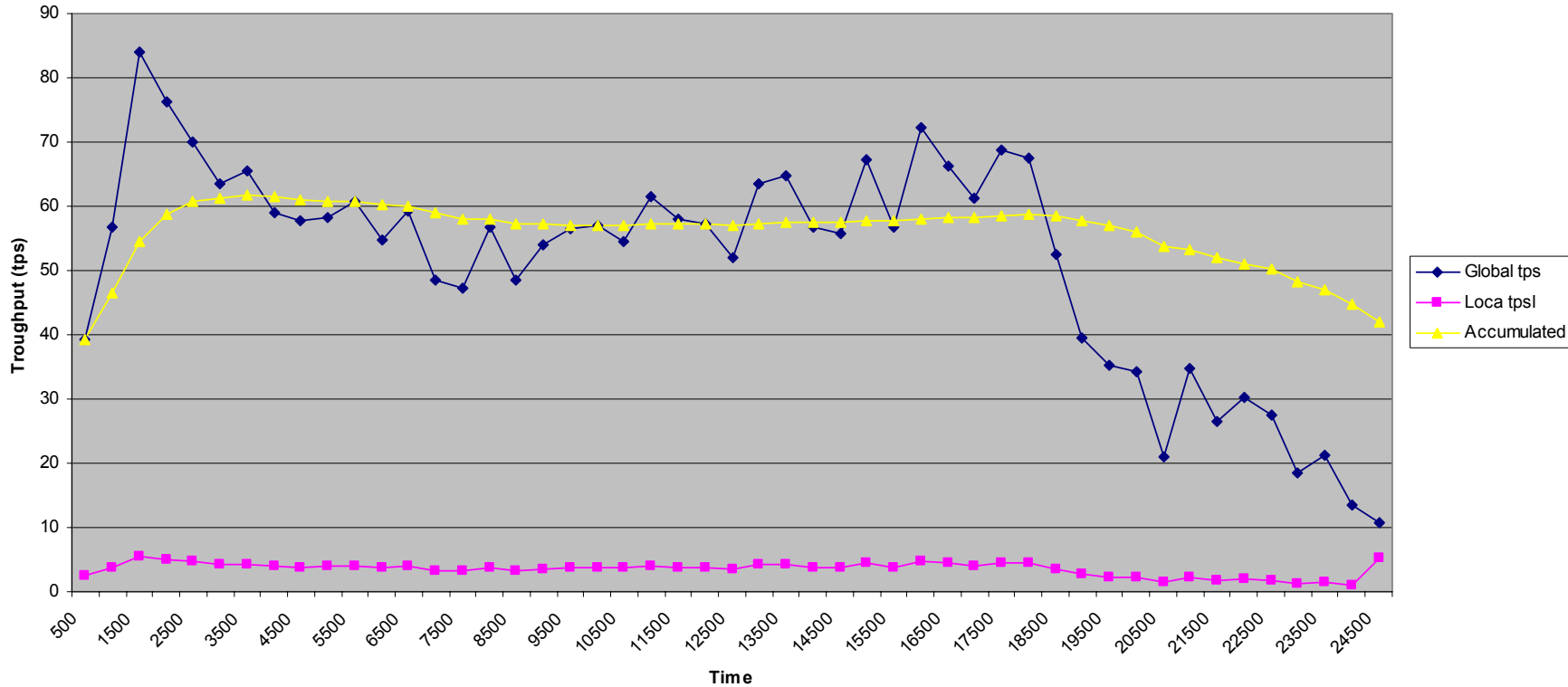


Initial results

- Software environment:
 - Java 1.4.2; globus 3.2.1 with OGSA container;
- Grid environment
 - Cluster with 20 pentium IV, 1.7 Ghz, 256MB RAM
 - Linux 2.4.20-31.9
- SVA application
 - 1000 particles;25 iterations for each particle;
 - Geometry 71732 tetrahedrons;25 time-instants

Throughput

Average node throughput



Related work

- OGSA-DAI
- Grid query processing systems:
 - OGSA-DQP(Polar*)
- Telegraph system
 - Eddy[sigmod00]
 - Statemodules[ICDE03]
 - ContinuousQueries[CIDR03]
 - Distributed Eddies[VLDB03]

Conclusion

- A parallel query processing system for grid;
 - Scheduling of available grid nodes;
 - Dynamic adaptation of tuple block size;
 - Re-evaluation of node allocation;
 - Support for iteration over QEP fragment;
 - Spatial-Time hash-join and user program invocation;
 - Up to 11 times faster wrt centralized solution
- Future work
 - Adaptivity model
 - Adapt spatial hash-join swap bucket strategies;
 - Decentralized multiple operations
 - Through validation;

Thank You !!!