Towards Real-Time, Many Task Applications on Large Distributed Systems

- focusing on the implementation of RT-BOINC

Sangho Yi (sangho.yi@inria.fr)
Content

- Motivation and Background
- **RT-BOINC** in a nutshell
  - Internal structures
  - Design & implementation
- Conclusions and future work
Motivation

- Demands for computing large-scale real-time (RT) tasks increased in distributed computing environment
  - Chess, Game of Go
  - Real-time Forensic Analysis
  - Ultra HD-level Real-time Multimedia Processing
  - ...

- Lack of support for RT in existing Desktop Grids, and Volunteer Computing environment
About BOINC

- BOINC is tailored for maximizing task throughput, not minimizing latency on the order of seconds.
  - Same as XtreemWeb and Condor

- A BOINC project has
  - A BOINC server (web, storage, database, ...)
  - Multiple BOINC clients
  - Network connection between server - clients
BOINC Projects

- Normally perform a few transactions in 1 sec with host clients.
  - 1~15 transactions in 1 sec (ref. http://boincstats.com)
- Send large chunk of computation to the host clients.
  - a couple of hours, or even days of computation
- Does not have RT guarantee
  - Because it is tailored for maximizing total amount of computation.
Significant Gaps here...

- “I need a 10-second-car.” - in the movie “Fast & Furious”

Vin diesel – the main actor in the movie
Significant Gaps here...

- "We need a 10-second-completion." - in a "Chess game"

Chess player’s mission: Get next move within 10 seconds

Volunteer computing server (ex> BOINC)

# of transactions $\cong$ 1,000 per second

Volunteer hosts (ex> BOINC hosts)

$\cong$ 10,000 hosts
RT-BOINC in a Nutshell

- RT-BOINC features
  - Providing low WCET (worst-case execution time) for all components
  - No database operations at run-time
  - O(1) interfaces for data structures
  - Reduced complexity for server daemons
    - Almost O(1)
BOINC Project

Requests for work distribution

BOINC Server

Work-generator

BOINC Hosts

Host

BOINC Host

BOINC Host

BOINC Host...

BOINC Host

BOINC Host

BOINC Host

BOINC Host

BOINC Host

Requests for work distribution

Work-generator

workunits in DB

w w w w w

w w w w w

Work-generator

workunit-results in DB

w w w w

w w w w

workunit-results in DB

w w w w

w w w w

workunit-results in DB

w w w w

w w w w

workunit-results in DB

w w w w

w w w w

workunit-results in DB

w w w w

w w w w

workunit-result ready queue

workunit-result ready queue

workunit-result ready queue

workunit-result ready queue

Validator

Assimilator

Scheduler

File-deleter

Feeder

: flow of distributing work requests

: flow of reporting work results
RT-BOINC Internal

RT-BOINC Project
- Requests for work distribution

RT-BOINC Server
- RT-Work-generator
  - workunits in O(1) data structures
  - workunit-results in O(1) data structures
- RT-Feeder
  - workunit-result ready queue

RT-BOINC Project
- RT-Transitioner
- RT-Assimilator
- RT-Validator
- RT-Scheduler

BOINC Hosts
- Host
- Host
- Host

: flow of distributing work requests
: flow of reporting work results
**Data management**

- MySQL Database vs. In-memory data structures

### Main Database

**BOINC DB**

(workunits, results, hosts, users, apps, platforms, and ...) - based on MySQL

Complexity for lookup, insert, and remove: \( O(\log N) \sim O(N^2) \)

(a) BOINC

### In-Memory Data structures - \( O(1) \)

- Multi-level lookup tables and fixed-size list

(b) RT-BOINC

- Lookup pools

In-memory data records with data format compaction (workunits, results, hosts, users, ...) - based on shm-IPC
Example 1) select from where;

- Retrieving RESULT from the O(1) data structure

Ex) select * from result where workunitid = '0x1234';
Example 2) insert into values(...);

- Inserting RESULT to the O(1) data structure

```
(Ex) insert into result ... values (...);
```

Get an available result field’s id from end of list
Then, remove the ‘id’ from end of list

```
|   |   |   |   |   |   |   |   |   |
```

Lookup pool for available results

Insert result to this place

(a) Insertion

Result table in main memory
Example 3) delete from where;

- Deleting RESULT from the O(1) data structure

Ex) delete from result where id='1234';

(b) Deletion

Insert ‘1234’ to the end of the result lookup list

Lookup pool for available results

Invalidate 1234\textsuperscript{th} result

Result table in main memory
Prototype Implementation

- Additional information
  - Compaction of BOINC's data format
  - Modification of PHP codes
  - Trade-offs between memory usage and WCET
    - Statically adjustable with parameters
  - Compatibility with BOINC
    - The rest parts are still compatible with BOINC.
Size of Data Structures

- RT-BOINC uses the ‘shared memory segment’ IPC between server daemon processes to share the data structures.

- For 10,000 entries of hosts, results, workunits, it consumes totally 1.09GB in main memory.
  - Memory overhead for O(1) data structures is 38.6% of the total usage.
  - Using 1GB memory is reasonable on the common-off-the-shelf 64-bit hardware platforms.
Detailed information on the Web


**RT-BOINC** stands for a Real-Time BOINC

It was designed for managing highly-interactive, short-term, and massively-parallel real-time applications. We designed and implemented RT-BOINC on top of BOINC server source codes.

Contact information: Sangho Yi and Derrick Kondo

---

**For users**

- Download RT-BOINC files
- Project detail and discuss
- Get support
- Donate money

**For developers**

- Join this project:
  To join this project, please contact the project administrators of this project, as shown on the [project summary page](#).

- Get the source code:
  Source code for this project may be available as [downloads](#) or through one of the SCM repositories used by the project, as shown on the [project page](#).
Source code on the Web

http://sourceforge.net/projects/rt-boinc
Performance Evaluation

- **Purpose**: to measure real-time performance of **BOINC** and **RT-BOINC**

- **Criteria**: the worst-case and the average execution time

- **Method**: micro and macro benchmarks
  - Micro-benchmark: for each primary operation related to server process
  - Macro-benchmark: for each server process (including feeder, scheduler, transitioner, work-generator, assimilator, validator, and file-deleter)
Experimental Environment

- We used a little bit slow, common-off-the-shelf system. ;-) 
  - For ease of reproduction of the results

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>1.60GHz, 3MB L2 cache</td>
<td>Intel Core 2 Duo</td>
</tr>
<tr>
<td>Main Memory</td>
<td>3GB (800 Mhz)</td>
<td>Dual-channel DDR3</td>
</tr>
<tr>
<td>Secondary Storage</td>
<td>Solid State Drive</td>
<td>SLC Type</td>
</tr>
<tr>
<td>Operating System</td>
<td>Ubuntu 9.10 (karmic)</td>
<td>Linux Kernel 2.6.31-19</td>
</tr>
<tr>
<td>BOINC version</td>
<td>Server stable version</td>
<td>Nov. 11, 2009 (from SVN)</td>
</tr>
</tbody>
</table>
Micro-benchmarks

- Average execution time (in seconds)

![Graph showing average execution time for various operations]
**Micro-benchmarks**

- Worst-case execution time (in seconds)

![Graph showing the comparison between BOINC and RT-BOINC operations](image)
Micro-benchmarks

- Performance improvement ratio (RT-BOINC / BOINC)
Micro-benchmarks

- Performance gap between worst-case and average
Macro-benchmarks (low load)
Macro-benchmarks (high load)
Macro-benchmarks

- Difference of worst-case performance between low and high load condition
Conclusions

- RT-BOINC provides...
  - 30~100 times higher average performance than BOINC.
  - 300~1,000 times lower WCET (worst-case execution time) for the given load condition.
  - less difference between the average and the worst-case performance.
  - less difference between low and high load conditions.
Future work (The rest part)

RT-BOINC server provides the worst-case number of transactions processing per second: \( N_t \)

Red: What we’ve done in the first paper

<table>
<thead>
<tr>
<th></th>
<th>Time for handling transactions in server</th>
<th>Time for computation in volunteer hosts</th>
<th>Time for communication between server and hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{N_c}{N_t} )</td>
<td>Time for handling transactions in server</td>
<td>Time for computation in volunteer hosts</td>
<td>Time for communication between server and hosts</td>
</tr>
</tbody>
</table>

Checkpointing & Replication is required in the presence of hosts’ failures.
Future work (Remaining issues)

- Providing ‘dynamic shared-memory management’ for reducing memory usage
- Studying trade-offs between time(WCET) and space(memory usage)
- Providing ‘full functionality’ including locality scheduling, and homogeneous redundancy
- Testing it with ‘real’ applications such as Chess, Game of Go, and etc.
Thanks! / Questions?