GrenchMark: A Framework for Analyzing, Testing, and Comparing Grids

A. Iosup, H. Mohamed, D.H.J. Epema
PDS Group, ST/ EWI, TU Delft

C. Dumitrescu (TU Delft/ U. Muenster),
I. Raicu, I. Foster (U. Chicago),
M. Ripeanu (UBC),
C. Stratan, M. Andreica, N. Tapus (UPB)
1. Introduction and Motivation

1.1. A Layered View of Grids (1/2)

- **Layer 1: Hardware + OS**
  - Automated
  - Simple
- **Layers 2-4: Grid Middleware Stack**
  - Low Level: file transfers, local resource allocation, etc.
  - High Level: grid scheduling
  - Very High Level: application environments (e.g., distributed objects)
  - Automated/user control
  - Simple to complex
- **Layer 5: Grid Applications**
  - User control
  - Simple to complex
1. Introduction and Motivation

1.1. A Layered View of Grids (2/2)

- **Unitary applications**
  - Single processor, single node
  - Parallel/distributed: MPI, Java RMI, Ibis, ProActive ...

- **Composite applications**
  - Bag of tasks
  - Chain of jobs
  - Direct Acyclic Graph-based Workflows
1. Introduction and Motivation

1.2. Complexity => More Testing Required

- Server: 99.99999% reliable
- Small Cluster: 99.999% reliable

**Today’s grids are complex and error-prone!**

- DAS-2: >10% jobs fail [Ios06]
- TeraGrid: 20-45% failures [Kha06]
- Grid3: 27% failures, 5-10 retries [Dum05]
1. Introduction and Motivation

1.2. Complexity => More Testing Required
(Grid reality worse than theory)

- NMI Build-and-Test Environment at U.Wisc.-Madison:
  112 hosts, >40 platforms (e.g., X86-32/Solaris/5, X86-64/RH/9)
- Serves >50 grid middleware packages: Condor, Globus, VDT, gLite, GridFTP, RLS, NWS, INCA(-2), APST, NINF-G, BOINC …

Two years of functionality tests (‘04-‘06): over 1:3 runs have at least one failure!

(1) Test or perish!
(2) In today’s grids, reliability is more important than performance!
Outline

1. Introduction and Motivation
2. Testing in Grids
3. The GrenchMark Framework
4. Experiments with GrenchMark
5. GrenchMark Success Stories
6. Conclusion
2. Testing in Grids

2.1. Testing for Design Adequacy

• Functional testing
  • Various application types: Ibis, workflows, bag-of-tasks, etc.
  • Delegated submissions

• Operational testing
  • A user cannot get >50% of the system for >1 hour
  • Monitoring data sampled every 15 minutes

• Scalability testing
  • Maximum utilization >95% for an hour
  • Utilization >75% over a month
2. Testing in Grids

2.2. Testing Performance

- Load testing
  - Wait time of jobs under heavy load
  - Detailed application performance

- Benchmarking
  - Number of items completed in 1 hour, or
  - Time to complete 1 item

- Scenario testing
  - Performance in a steady system with Poisson arrivals
  - Performance of two systems, when combined
2. Testing in Grids

2.3. Testing Reliability

- Reliability growth testing
  - Increase reliability after each iteration

- Burn-out testing
  - Systems have “bath-tub reliability curve”
  - Users get a tested system (after burn-out)

- System integration testing
  - Is the sum of reliable components reliable?
2. Testing in Grids

2.4. Analyzing, Testing, and Comparing Grids

- Use cases for automatically analyzing, testing, and comparing grids (or grid middleware)
  - Functionality testing and system tuning
  - Performance testing/analysis of grid applications
  - Reliability testing of grid middleware
  - …

- **For grids, this problem is hard!**
  - Testing in real environments is difficult
  - Grids change rapidly
  - Validity of tests
  - …
Outline

1. Introduction and Motivation
2. Testing in Grids
3. The GrenchMark Framework
4. Experiments with GrenchMark
5. GrenchMark Success Stories
6. Future Work
7. Conclusions
3. The GrenchMark Framework
3.1. GrenchMark: Generic (Grid) Testing

- What’s in a name? 
  grid benchmark → working towards a generic tool for the whole community: help standardizing the testing procedures, but benchmarks are too early; we use synthetic grid workloads

GrenchMark can generate and run synthetic grid workloads

- A set of metrics for analyzing grid settings
- A set of representative grid applications
  - Both real and synthetic
- Easy-to-use tools to create synthetic grid workloads
- Flexible, extensible framework
3. The GrenchMark Framework
3.2. GrenchMark Overview: Easy to Generate and Run Synthetic Workloads
3. The GrenchMark Framework
... but More Complicated Than You Think

• Workload structure
  • User-defined and statistical models
  • Dynamic jobs arrival
  • Burstiness and self-similarity
  • Feedback, background load
  • Machine usage assumptions
  • Users, VOs

• Metrics
  • A(W) Run/Wait/Resp. Time
  • Efficiency, MakeSpan
  • Failure rate [!]

• Notions
  • Co-allocation, interactive jobs, malleable, moldable, …

• Measurement methods
  • Long workloads
  • Saturated / non-saturated system
  • Start-up, production, and cool-down scenarios
  • Scaling workload to system

• Applications
  • Synthetic
  • Real

• Workload definition language
  • Base language layer
  • Extended language layer

• Other
  • Can use the same workload for both simulations and real environments
3. The GrenchMark Framework
3.3. Workload Generation (1/2)

- **Trace-based**
  - Replay user requests
  - Data: Grid Workloads Archive, Parallel Workloads Archive

- **Model-based**
  - Mathematical distributions for workload characteristics
  - Use parameters from traces (!)
  - Data: Hui Li for grid, Lublin-Feitelson for HPC

- **User-specified**
  - New or Hybrid models
  - Mixing of traces
3. The GrenchMark Framework

3.3. Workload Generation (2/2)

- Workload description file, simple format + extensions

**Simple format:**

```plaintext
# File-type: text/wl-spec
#NJobs Composition Type SiteType Compo SiteInfo ArrivalTimeDistr OtherInfo
25 composite DAG single 1 *:? Poisson(120s) StartAt=0s
25 unitary sserio co-alloc 5 *:? C(120s) StartAt=50s, ExternalFile=1.xin
25 unitary smpil single 1 *:? C(120s) StartAt=30s, ExternalFile=2.xin
25 unitary smpil single 1 *:? C(120s) StartAt=90s, ExternalFile=3.xin
```

- Number of jobs
- Co-allocation and parallelism
- Combining four workloads into one
- First job and job arrival time distribution

May 7, 2007
3. The GrenchMark Framework
3.4. Workload Submission

• Submission (mostly) decoupled from workload generation

• Submission engine architecture
  • Multiple threads (good for multi-core systems)
  • Hierarchical (when using ServMark)

• Open vs. Closed submission
  • Open = new jobs arrive independently of previous jobs’ completion
  • Closed = use system feed-back to start jobs, e.g., chain of tasks, workflows
  • Also Partially-Open (only some jobs have feed-back)
3. The GrenchMark Framework
3.5. Data Collection and Analysis (1/2)

• Metrics of interest (sample)
  • Testing design adequacy
    • Ratio of jobs completed from started
    • Fairness
  • Testing performance
    • Time-related: wait time, run time, response time
    • Administrator: utilization, number of jobs completed, number of users served, middleware overhead
    • User: application makespan, number of results, middleware overhead, throughput
  • Testing reliability
    • MTTFailure, MTTRepair, hazard rate function (shape)
3. The GrenchMark Framework

3.5. Data Collection and Analysis (2/2)

• Reporting
  • Basic statistics
    • Mean, Min, Max, Std.Dev., Median, Quantiles

• Higher moments
  • Kurtosis, Skewness

• Distributions
  • Empirical distributions (histograms, CDF plot)
  • Curve fitting and goodness-of-fit for well-known distributions: Normal, Lognormal, Weibull, Exponential, etc.
3. The GrenchMark Framework
3.6. GrenchMark’s Current Status

- Already done in Python [http://www.python.org]
  - Workload Generator
  - Generic Workload Submitter (Koala, Globus GRAM, Condor, option for JSDL, PBS, LSF, SGE, …)
  - Results analyzer (crude)
  - Applications
    - Unitary, 3 types: sequential, MPI, Ibis (Java)
    - +35 applications
    - Composite applications: DAG-based

- Ongoing work
  - Automated results analyzer
  - Experiments: test, test, test…
Outline

1. Introduction and Motivation
2. Testing in Grids
3. The GrenchMark Framework
4. Experiments with GrenchMark
5. GrenchMark Success Stories
6. Future Work
7. Conclusions
4. Experiments with GrenchMark

4.1. Testing Design Adequacy

- **System functionality testing**: show the ability of the system to run various types of applications
  - Report failure rates

  [10% job failure rate in a controlled system like the DAS!]

<table>
<thead>
<tr>
<th>Workload</th>
<th>Types of applications</th>
<th># of CPUs</th>
<th>Component no.</th>
<th>size</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmark+</td>
<td>synthetic, seq. &amp; MPI</td>
<td>1-128</td>
<td>1-15</td>
<td>1-32</td>
<td>85%</td>
</tr>
<tr>
<td>ibis+</td>
<td>various, Ibis</td>
<td>2-32</td>
<td>1-8</td>
<td>2-16</td>
<td>65%</td>
</tr>
<tr>
<td>unitary</td>
<td>gmark+ &amp; ibis+</td>
<td>1-32</td>
<td>1-8</td>
<td>1-32</td>
<td>90%</td>
</tr>
</tbody>
</table>

- **Periodic system testing**: evaluate the current state of the grid
  - Replay workloads
4. Experiments with GrenchMark
4.2. Testing Performance (1/2)

- **Testing application performance**: test the performance of an application (for sequential, MPI, Ibis applications)
  - Report runtimes, waiting times, grid middleware overhead
  - Automatic results analysis

<table>
<thead>
<tr>
<th>Job name</th>
<th>Job type</th>
<th>Turnaround [s]</th>
<th>Runtime [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sser</td>
<td>seq</td>
<td>129</td>
<td>44</td>
</tr>
<tr>
<td>mpi1</td>
<td>MPI</td>
<td>332</td>
<td>110</td>
</tr>
<tr>
<td>NQueens</td>
<td>Ibis</td>
<td>99</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Job name</th>
<th>Turnaround Range</th>
<th>Runtime Range</th>
<th>Run</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>sser</td>
<td>16-926</td>
<td>1-588</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>mpi1</td>
<td>21-1078</td>
<td>1-332</td>
<td>80%</td>
<td>81%</td>
</tr>
<tr>
<td>NQueens</td>
<td>15-1835</td>
<td>1-201</td>
<td>70%</td>
<td>85%</td>
</tr>
</tbody>
</table>

- **What-if analysis**: evaluate potential situations
  - System change
  - Grid inter-operability
  - Special situations: spikes in demand
4. Experiments with GrenchMark
4.3. Testing Performance (2/2)

• **Single-site vs. co-allocated jobs:**
  *compare the success rate of single-site and co-allocated jobs, in a system without reservation capabilities*
  *Single-site jobs 20% better vs. small co-allocated jobs (<32 CPUs), 30% better vs. large co-allocated jobs [setting and workload-dependent !]*

• **Unitary vs. composite jobs:**
  *compare the success rate of unitary and composite jobs, with and without failure handling mechanisms*
  *Both 100% with simple retry mechanism [setting and workload-dependent !]*
4. Experiments with GrenchMark

4.4. Testing Reliability

• Testing a 1500-processors Condor environment: what is the success rate of single-processor jobs when submitted in batches of various sizes?
  - Workloads of 1000 jobs, grouped by 2, 10, 20, 50, 100, 200
  - Goal: finish jobs in at most 1h after the last submission
  - Results
    • >150,000 jobs submitted
    • >100,000 jobs successfully run, >2 yr CPU time in 1 week
    • 5% jobs failed (much less than other grids’ average)
    • 25% jobs did not start in time and were cancelled
Outline

1. Introduction and Motivation
2. Testing in Grids
3. The GrenchMark Framework
4. Experiments with GrenchMark
5. GrenchMark Success Stories
6. Conclusions
5. GrenchMark Success Stories
5.1. Releasing the Koala Grid Scheduler on the DAS

- **Koala** [http://www.st.ewi.tudelft.nl/koala/](http://www.st.ewi.tudelft.nl/koala/)
  - Grid Scheduler with co-allocation capabilities
- **DAS: The Dutch Grid, ~200 researchers**
- **Initially**
  - Koala, a tested (!) scheduler, pre-release version
- **Test specifics**
  - 3 different job submission modules
  - Workloads with different jobs requirements, inter-arrival rates, co-allocated v. single site jobs…
  - Evaluate: job success rate, Koala overhead and bottlenecks
- **Results**
  - 5,000+ jobs successfully run (all workloads); functionality tests
  - 2 major bugs first day, 10+ bugs overall (all fixed)
  - **KOALA is now officially released on the DAS**
    (full credit to KOALA developers, 10x for testing with GrenchMark)
5. GrenchMark Success Stories

5.2. The LittleBird Peer-to-Peer protocol

- **LittleBird, part of Tribler** [http://www.tribler.org](http://www.tribler.org]
  - Tribler – a Peer-to-Peer file-sharing system, >100,000 downloads/year
  - LittleBird – an epidemic protocol that exchanges swarm information between peers that are currently or were recently members of a swarm

- **Test specifics**
  - Large-scale test environment based on GrenchMark
  - Workload: peer arrival / departure
  - Evaluate: job success rate, Koala overhead and bottlenecks

- **Results (by Jelle Roozenburg)**
  - Functionality proof
  - Protocol is scalable, resilient against DoS and pollution attacks
  - **LittleBird is going to be integrated into Tribler**
5. GrenchMark Success Stories

5.3. The ServMark Grid Services Tester

ServMark is a hierarchical grid services tester, and part of the Globus Incubator Project

- ServMark is a Globus Incubator Project
  [http://dev.globus.org/wiki/Incubator/ServMark]
  - Functionality proof: test HTTP servers
  - First release in October 2006
  - New release scheduled for Q4 2007
Outline

1. Introduction and Motivation
2. Testing in Grids
3. The GrenchMark Framework
4. Experiments with GrenchMark
5. GrenchMark Success Stories

6. Conclusions
Conclusion

Today’s grids are complex and error-prone

In today’s grids, reliability is more important than performance, so:
Test or perish!

GrenchMark can generate and run synthetic grid workloads

ServMark is a hierarchical grid services tester
Thank you!

Questions? Remarks? Observations?

GrenchMark
http://grenchmark.st.ewi.tudelft.nl/ [10x Paulo]

Alexandru IOSUP
TU Delft
A.Iosup@tudelft.nl

May 7, 2007