TAUdb: PerfDMF Refactored

Kevin Huck, Suzanne Millstein, Allen D. Malony and Sameer Shende
Department of Computer and Information Science
University of Oregon
PerfDMF Overview

- Performance Data Management Framework
- Started in 2004 (Huck et al., ICPP 2005)
- Database Schema & Java API (profile parsing, database queries, conversion utilities)
- Provides DB support for TAU Profile Analysis Tools ParaProf, PerfExplorer, EclipsePTP
- Used as regression testing database for TAU
- Used as performance regression database for FACETS (2008-2012)
- Ported to several DBMS: PostgreSQL, MySQL, H2, Derby, Oracle, DB2
Supported Profile Formats

- **TAU** – profiles, packed profiles, snapshots (UO)
- **DynaProf** – PAPI DynaProf profiles (UTK)
- **mpiP** – Lightweight, scalable MPI Profiling (Vetter, Chambreau)
- **HPM Toolkit** profiles (IBM)
- **Gprof** profiles (GNU)
- **PerfSuite** psrun profiles (NCSA)
- **Cube, Cube3, Cube4** profiles (FZJ)
- **HPC Toolkit** profiles (Rice)
- **OMPP** – OpenMP Profiler profiles (Fuerlinger)
- **PERI-XML** (PERI)
- **GPTL** – General Purpose Timing Library profiles (ORNL)
- **Paraver** profiles (BSC)
- **IPM** – Integrated Performance Monitoring (NERSC)
- **Google** profiles (Google)
- **Others** (Gyro, GAMESS, other application-specific timer data)
PerfDMF Schema Overview

Application → Experiment → Trial

Timer → Metric → Counter

Measurement → Measurement

Nice and simple, but there are problems...
A Little CScADS History…

- Meeting: “Performance Tools for Petascale Computing”
  July 21-24, 2008

- PERI-XML Working Group: Towards a Common Exchange Format

- Main Focus: Each data point has a connection to Five profile dimensions
  - Code (static location, binary/source) – Check!
  - Space (physical and logical) – Check!
  - Metrics (data which is collected, derived values) – Check!
  - Dynamic State (callstack, context, …) – um, not explicitly
  - Time (timeline) – no
PerfDMF Complaints Problems “Challenges”

- Metadata is not a first-class citizen in schema
  - Compressed XML document, not context-sensitive
- Hierarchy inadequate (too many/few levels)
- Not enough *explicit* semantic relationships in the data
  - Callpaths, phases, timer groups, etc.
- No *explicit* support for “special cases”
  - Callpaths, phases, parameters*, snapshots/timestamps
- Inefficiencies
  - Space (some poor normalization)
  - Time (parsing XML slow, loading big trials slow, etc)
- No C API
TAUdb – PerfDMF, but better!

- New Schema – (hopefully) all problems addressed
- TAU Tools retain compatibility with old schema
- Refactoring Java API
- Redesigned TAU measurement interface
- Full-featured C API
- SQLite support (evaluating)
TAUdb Schema Organizational Change

- **Application, experiment tables are gone**
- **View** (user-defined grouping/filter of Trials or Views of Trials – arbitrary depth) defined using Metadata
  - Replacement for application & experiment
- **Trial** (still a single profile)
  - **Primary_Metadata** – name : value pair, common to all threads, no hierarchical data – handles most common cases
  - **Secondary_Metadata** – could be unique for each thread, phase, timer, can be hierarchical or arrays
  - Both can be queried directly
    - View: “Give me trials of application ABC with 4096 processes which ran on machine X with dataset Y in the last 30 days”
View Creation Process

- Using trial metadata name/values, filter (out) the trials of interest
- Same interface in PerfExplorer, ParaProf, web

View creation GUI mock-up
Metadata Collection

- Like PerfDMF, can load Metadata file with profile data
  - XML
  - JSON [http://www.json.org](http://www.json.org)
    - Like XML, allows for arbitrarily structured data
    - Less annotation overhead, no pre-defined schema

- Example:

  ```json
  { "metadata_number" : 14,
    "metadata_string" : "string",
    "metadata_boolean" : false,
    "metadata_array" : [1,2,3],
    "metadata_null" : null,
    "metadata_object" : { "inner_object" : "value" },
    "metadata_array_of_objects" : [ { "name" : "value" }, { "next" : "value" } ] }  
  ```
JSON Format

- JavaScript Object Notation

Figure credits: http://www.json.org

Profile Data Schema Changes

- Redundancies (source location, calls, subroutines, node/context/thread)
- No explicit tables representing “location”, “context” or “state”
- Encoded strings (main => foo => iteration12 => bar)
- Only 3 of 5 dimensions explicitly supported
Simple Example

void bar (int x) {
    sleep(1);
}
void foo (int x) {
    sleep(1);
    bar(x);
}
int main (int argc, char** argv) {
    int x = 0;
    for (x = 0 ; x < 10 ; x++) {
        foo (x);
        bar (x);
    }
}

- What happens when we store a callpath profile of this program?
Simple example in old schema (reduced detail)

# rows = callgraph nodes + callgraph edges

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>File</th>
<th>Line</th>
<th>Line end</th>
<th>Column</th>
<th>Column end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>main()</td>
<td>test.c</td>
<td>12</td>
<td>18</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>foo()</td>
<td>test.c</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>main() =&gt; foo()</td>
<td>test.c</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>bar()</td>
<td>test.c</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>main() =&gt; foo() =&gt; bar()</td>
<td>test.c</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>main() =&gt; bar()</td>
<td>test.c</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

# rows = (callgraph nodes + callgraph edges) * # threads * # metrics

<table>
<thead>
<tr>
<th>Timer</th>
<th>Node</th>
<th>Thread</th>
<th>Metric</th>
<th>Inclusive</th>
<th>Exclusive</th>
<th>Incl %</th>
<th>Excl %</th>
<th>Calls</th>
<th>Subr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30003073</td>
<td>49</td>
<td>100.00%</td>
<td>0.00%</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20001985</td>
<td>10001026</td>
<td>66.67%</td>
<td>33.33%</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20001985</td>
<td>10001026</td>
<td>66.67%</td>
<td>33.33%</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20001998</td>
<td>20001998</td>
<td>66.67%</td>
<td>66.67%</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10000959</td>
<td>10000959</td>
<td>33.33%</td>
<td>33.33%</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10001039</td>
<td>10001039</td>
<td>33.33%</td>
<td>33.33%</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Profile Data Schema Changes

- Redundancies eliminated
- Thread table represents location
- TimerParam table represents state
- Callpath object represents call tree context
- Normalizing the schema results in space savings (timer, metric) - ~30%
- Long names eliminated
### Simple Example in New Schema

**TIMER**: # rows = callgraph nodes

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>File</th>
<th>Line</th>
<th>Line end</th>
<th>Column</th>
<th>Column end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>main()</td>
<td>test.c</td>
<td>12</td>
<td>18</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>foo()</td>
<td>test.c</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>bar()</td>
<td>test.c</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**CALLPATH**: # rows = callgraph nodes + callgraph edges

<table>
<thead>
<tr>
<th>ID</th>
<th>Timer</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**THREAD**: # rows = # threads

<table>
<thead>
<tr>
<th>ID</th>
<th>Node</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**CALLPATH_DATA**: # rows = (callgraph nodes + callgraph edges) * # threads

<table>
<thead>
<tr>
<th>Callpath</th>
<th>Thread</th>
<th>Calls</th>
<th>Subr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

**MEASUREMENT**: # rows = (callgraph nodes + callgraph edges) * # threads * # metrics

<table>
<thead>
<tr>
<th>TCD Metric</th>
<th>Incl</th>
<th>Excl</th>
<th>Incl %</th>
<th>Excl %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>130003073</td>
<td>49</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>2</td>
<td>120001985</td>
<td>10001026</td>
<td>66.67%</td>
<td>33.33%</td>
</tr>
<tr>
<td>3</td>
<td>120001985</td>
<td>10001026</td>
<td>66.67%</td>
<td>33.33%</td>
</tr>
<tr>
<td>4</td>
<td>120001998</td>
<td>20001998</td>
<td>66.67%</td>
<td>66.67%</td>
</tr>
<tr>
<td>5</td>
<td>110000959</td>
<td>10000959</td>
<td>33.33%</td>
<td>33.33%</td>
</tr>
<tr>
<td>6</td>
<td>110001039</td>
<td>10001039</td>
<td>33.33%</td>
<td>33.33%</td>
</tr>
</tbody>
</table>
TAUdb redesigned schema

✓ Metadata can be queried, context-sensitive
✓ All 5 dimensions now explicit

- Trial
- Timer
- Callpath
- CallpathData
- Interval
- Metric
- Measurement
- Thread
- TMParam
- Metadata
- Counter
- Measurement
**C API**

- C programming interface under development
- PostgreSQL support first, others as requested
- Prototype developed
  - Query only, both old and new schema
- Plan full-featured API: Query, Insert, & Update
- One internal test user so far – Nick Chaimov using it for Active Harmony / CHiLL work
- Request for SQLite support – currently evaluating JDBC clients
Project Status, Conclusions, Future

- New database schema defined
- C API in development
- Java API (mostly) supports the new schema
  - Supports JSON metadata (and previous XML support)
  - Not in most TAU recent release
- View construction GUI still in design phase
- TAU measurement API needs design, implementation, testing (context-sensitive metadata)
- Planning targeted distribution
  - TAUdb, ParaProf, PerfExplorer
  - Goal: zero-step config (likely 1-step config)
Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
  - ASC/NNSA
  - SUPER project
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Research Centre Juelich
- Argonne National Laboratory
- Technical University Dresden
- ParaTools, Inc.
- NVIDIA