PerfTrack

A Performance Data Storage and Analysis Tool
Steps for Using PerfTrack

1. Gather Machine Data
2. Build Application
3. Execute Application
4. Load Data
5. Analyze Data

Build Application
Execute Application
Simple Analysis

Gather Machine Data
Build Data
Environment Data
Performance Data
Machine Data

DBMS

105% Faster!
72% Slower
Steps for Using **PerfTrack**: Gather Machine Data

1. Gather Machine Data
2. Build Application
3. Execute Application
4. Load Data
5. Analyze Data

- Build Application
- Execute Application
- Simple Analysis
- Build Data
- Environment Data
  - Performance Data
- DBMS
- Gather Machine Data
- Machine Data
- 105% Faster!
- 72% Slower
Gathering machine characteristic data for PerfTrack is currently a manual process. Here are the characteristics that we currently collect for the machines in our PerfTrack installation. The figures above represent PerfTrack resources. The blue bubble shows the type of the resource. The yellow bubble contains the descriptive attributes we collect for that type of resource. We model machines with a resource hierarchy with a root of grid. A grid may have one or more machine nodes as children. A machine can be broken up into one or more partitions. The children of partitions are the nodes that belong to the partitions. The children of nodes are processors.
Steps for Using *PerfTrack*: Build Application

1. Gather Machine Data
2. **Build Application**
3. Load Data
4. Analyze Data

- Build Application
- Execute Application

- Gather Machine Data
- Machine Data
- Build Data
- Environment Data
- Performance Data

DBMS

- 105% Faster!
- 72% Slower
Build Application

PerfTrack has a script interface to automatically build and gather build-time information about the application. The PerfTrack build command is `ptbuild.py`. We show an example of its use below.

```
ptbuild.py --app umt2k --pathToExe ./umt2k --srcDir . -V
```

This collects information about the build, such as the machine the application was built on, the compilers used, and environment variables that were set during the build.
Steps for Using *PerfTrack*: Execute Application

1. Gather Machine Data
2. Build Application
3. **Execute Application**
4. Load Data
5. Analyze Data

- **Build Application**
- **Execute Application**
- **Simple Analysis**

- **Gather Machine Data**
- **Machine Data**
- **Build Data**
- **Environment Data**
  - Performance Data

- **DBMS**

Evaluation:
- 105% Faster!
- 72% Slower
Execute Application

PerfTrack also has scripts to capture information about the runtime environment and execution of the application. The PerfTrack command to execute the application is ptrun.py. We show an example of its use below.

```
ptrun.py --app umt2k --batchFile psub.script --inputDeck "opacfile,rtin,smartin" --exeName ./umt2k
```

This collects information about the execution, such as the machine the application was run on, and environment variables that were set during the execution.
Execute Application

PerfTrack has parsers for extracting performance data from files after the execution is complete. PerfTrack currently has parsers for several ASCI Purple Benchmarks as well as for several performance tools. During the parsing of the performance data, we gather data about the performance tools used, the metrics for which data was gathered, possible time intervals defined in the execution, and create performance results for the data reported.
Steps for Using *PerfTrack*: Load Data

1. Gather Machine Data
2. Build Application
3. Execute Application
4. Load Data
5. Analyze Data

**Diagram:**
- **DBMS**
  - Gather Machine Data
  - Machine Data
  - Build Data
  - Environment Data
  - Performance Data

**Performance Statistics:**
- 105% Faster!
- 72% Slower
Load Data

PerfTrack has scripts that load the collected data in PerfTrack data format (PTdf) into the database. PTdf is a simple API for defining the resources, resource types, and performance results for entry into the database. There are eight types of PTdf statements. The specification for these statements is shown below.

**PTdf - PerfTrack Data Format**

- **Application** appName
- **ResourceType** resourceName resourceTypeName
- **Execution** execName appName
- **Resource** resourceName resourceTypeName execName
- **Resource** resourceName resourceTypeName
- **ResourceAttribute** resourceName attributeName attributeValue attributeType
- **PerfResult** execName resourceSet perfToolName metricName value units startTime endTime
- **ResourceConstraint** resourceName1 resourceName2
Load Data: PTdf Examples

Here are some examples of PTdf statements for a real execution in our PerfTrack installation.

**Resource Type definitions**
- ResourceType application
- ResourceType build
- ResourceType execution

**Define an application resource**
Application /umt2k

**Define an execution and its attributes**
Execution /umt2k-1-1713 /umt2k
ResourceAttribute /umt2k-1-1713 ExeSize 1477477 string
ResourceAttribute /umt2k-1-1713 ExePerms 0755 string

**Define a build and its attributes**
Resource /build-1714 build /umt2k-1-1713
ResourceAttribute /build-1714 BuildNode mcr37 string
ResourceAttribute /build-1714 BuildEnvMALLOC_MMAP_MAX_ 0 string
Define a performance result

**PerfResult** /umt2k-1-1713 "/umt2k,/build-1714,/mpiicc-8.0 Version 8.0-1716,/mpiifort-8.0 Version 8.0-1717,/env-1719,/umt2k-1-1713,/SingleMachineMCR/MCR,/opacfile-1723,/rtin-1721,/smrtin-1722,/Linux #1 SMP Tue Jul 5 08:54:47 PDT 2005 2.4.21-p4smp-89chaos-1715,/submission-1720,/whole execution" "/self instrumentation" "/CPU time total" 8.1661E+01 min noValue noValue
Steps for Using **PerfTrack**: Analyze Data

1. Gather Machine Data
2. Build Application
3. Execute Application
4. Load Data
5. Analyze Data

- **DBMS**
  - **Gather Machine Data**
  - **Build Application**
  - **Execute Application**
  - **Analyze Data**
    - **Simple Analysis**
      - 105% Faster!
      - 72% Slower

**Build Data**, **Environment Data**, and **Performance Data** flow into the DBMS, which is connected to a computer displaying analysis results.
Analyze Data: Log into Database

PerfTrack has a GUI interface for analyzing the stored data. This is a screenshot of the PerfTrack login dialog.
Here we show the PerfTrack query window. We select three executions of the UMT2K benchmark for comparison. Each of these was executed on LLNL’s MCR cluster with 2 MPI processes. The execution umt2k-11-1933 has both processes running on the same node; umt2k-21-1823 has a single process per node; umt2k-1-1713 has a single process per node and 2 Open MP threads per process.
Analyze Data: Default Data Returned

This is the default set of data returned by PerfTrack. As you can see, we can't differentiate the executions with only this data. PerfTrack makes it possible to add data columns to this set.
Analyze Data: Add Data Fields

Here we add data columns to the default set of data returned by PerfTrack. The Resources window shows resources that are different for the executions. We are interested in differences in the execution attributes. We add the execution name and the NumberOfProcesses, ProcessesPerNode, and ThreadsPerProcess attributes.
Analyze Data: With Added Data Fields

Here is the PerfTrack data window with the added data fields. Now it is possible to differentiate the executions based on their runtime configurations.
Analyze Data: Create a Plot

Now, we would like to compare the performance data we have for these executions. First, we sort the data by metric name by clicking in the column header, “metric.” Then, we select the data values for the metric “Angle Loop Only time.” Then, we click the chart button, and make the X-axis column be the execution name, and the data label be the metric name. We name the plot “umt2k config” and click the “Create Plot” button.
Here is the plot generated by PerfTrack for the “Angle Loop Only time” metric. Please note: “Angle Loop Only time” for these runs was a CPU time measurement. Normally, this metric is a wall clock time measurement.
Now we add the data for the metric “CPU time total” to the chart. We select the appropriate data values and click “Add to Plot.”
Analyze Data: Plot with Data Added

Here is the chart generated by PerfTrack after we added the data for the metric “CPU time total.”
Analyze Data: Add Data to Plot

Here we add the data for the metric “Wallclock time total” to the plot.

<table>
<thead>
<tr>
<th>start_time</th>
<th>end_time</th>
<th>value</th>
<th>units</th>
<th>metric</th>
<th>tool</th>
<th>application</th>
<th>execution</th>
<th>NumberOfProcess</th>
<th>ProcessesPerNode</th>
<th>ThreadsPerProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>76.463</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>71.417</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-11-1933</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>61.426</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-21-1923</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>81.961</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-1-1713</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>75.585</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-11-1933</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>94.935</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-21-1923</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0.35409</td>
<td>min</td>
<td>Communication time</td>
<td>perftrack</td>
<td>jumtk-1-1713</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0.55901</td>
<td>min</td>
<td>Communication time</td>
<td>perftrack</td>
<td>jumtk-11-1933</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>0.34351</td>
<td>min</td>
<td>Communication time</td>
<td>perftrack</td>
<td>jumtk-21-1923</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0.00022204</td>
<td>min</td>
<td>Convergence time</td>
<td>perftrack</td>
<td>jumtk-1-1713</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0.00010181</td>
<td>min</td>
<td>Convergence time</td>
<td>perftrack</td>
<td>jumtk-11-1933</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0.00021</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-1-1713</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>42.024</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-21-1923</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>75.893</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-11-1933</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>61.756</td>
<td>min</td>
<td>Wallclock time total</td>
<td>perftrack</td>
<td>jumtk-21-1923</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Here is the plot with the data for three metrics displayed. From this, we can compare the performance of the executions. For example, execution “umt2k-1-1713” has the lowest wall clock time overall, and uses Open MP threading. From this we could conclude that for this particular configuration, application, and machine, using Open MP is advantageous.