Building a Big Data Chapel

Chris Taylor

DoD
Overview

• Big Data?
• Chapel on Mesos
• libhdfs3
• Machine Learning
• Current Projects
Big Data?

“Software, systems, and runtimes supporting – at minimum – resilient database style operations and features at scale.”
Chapel on Mesos
Chapel on Mesos

• What is Mesos?
  – Cluster/Cloud orchestration technology
  – Event/Actor/CSP communication model
    • Uses futures, options, and libevent/libev
  – cgroup containers
    • Specially identified pid_t's operating under kernel-level resource isolation
  – Emphasizes multi-tenancy, over-subscription
Chapel on Mesos

• Definitions
  – Mesos-Agents
Chapel on Mesos

• Definitions
  – Mesos-Agents
  – Mesos-Master(s)
Chapel on Mesos

• Definitions
  – Mesos-Agents
  – Mesos-Master(s)
  – Mesos-Framework
    • Executor
    • Scheduler
Chapel on Mesos

• Frameworks can be general or technology specific
  – General deployment solution
    • Aurora, Marathon, Chronos
  – Technology-specific deployment
    • Myriad (Hadoop-Yarn), Spark, Hadoop, MPI, Chapel
Chapel on Mesos

- Built a Mesos Scheduler for Chapel
  - User-friendly, integrates w/GASNET Customized Spawning
  - GASNET feature request
  - Consistently handles <= 32 tasks “well”
    - Greedy “task packing”
Chapel on Mesos

• Next work?
  – Needs a Customized Executor!
    • Handling task start-up issues
    • Exponential back-off
    • Core binding
  – Needs deployment hints added to Scheduler!
  – Mesos-Agents need CPU Isolation**
Chapel on Mesos

• Thank you to GASNET team
  – For providing the new Custom Spawning feature!
Chapel HDFS Support
libhdfs

• Apache's libhdfs
  – C wrapper library for Java Hadoop jars
  – This complicates life for Mesos users
    • Mesos “sandbox” needs libjvm.(so/a) and Hadoop jars
    • Deploy using Docker images?
      – Several hundreds of megabytes or gigabyte images
libhdfs3

• PivotalHD
  – libhdfs3 rooted in the native-hadoop project
  – C++ implementation of HDFS protocol for client applications
  – Deployment complications gone!
    • New complications related to HDFS deployment configuration!
libhdfs3

• Chapel runtime
  − Very approachable and well organized
  − Moving between Chapel code and the runtime was easy
  − Runtime's io system “plugin-like” design
  − ~1-2 weeks to get something working**
  − Took a couple months on/off again work to debug and tune

** Working != perfect
libhdfs3

- libhdfs3 now an CHPL_AUX_IO option in the runtime's io system!
  - Thank you Chapel team for sheparding!

- Next?
  - GlusterFS support
    - Avoid cgroup container access to FUSE
    - Initial version complete
    - Needs testing
Machine Learning with Chapel
Machine Learning

• Implemented
  – RandomForest (C++/Chapel)
  – Stochastic Logistic Regression (Python/Chapel)
  – Latent Dirichlet Allocation (Octave/Chapel)

• Measuring training time!

• Execution Environment
  – Amazon EC2 node
  – Chapel 0.13
    • jemalloc
    • qthreads
    • hwloc
  – CHPL_FLAGS=--fast --vectorize
Machine Learning

- Removed from evaluation
  - RandomForest (C++/Chapel)

- 0.13 compiler caught use of undocumented features the 0.12 compiler permitted
  - Specifically domain-related
  - Implementation heavily leveraged the undocumented features :(
  - Not enough time to fix the spaghetti code's issues
Machine Learning

• Stochastic Logistic Regression

• Data set?
  – MNIST training data – hand-written numbers, {0..9}
  – Samples have 784 features

• Left of Slide Graph – Stratified samples (sklearn)
  • Label 5 - 25000 samples
  • Label 6 - 20000 samples
  • Label 7 - 15000 samples
  • Label 8 - 10000 samples
  • Label 9 – 5000 samples

• Right of Slide Graph - All training samples
  • 50000 per Label
Machine Learning
Model Training

# Examples

<table>
<thead>
<tr>
<th># Examples</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25000</td>
<td>8</td>
</tr>
<tr>
<td>20000</td>
<td>6</td>
</tr>
<tr>
<td>15000</td>
<td>5</td>
</tr>
<tr>
<td>10000</td>
<td>4</td>
</tr>
<tr>
<td>5000</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Chapel**
- **Python**

Labels

<table>
<thead>
<tr>
<th>Labels</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Digit</td>
<td>18</td>
</tr>
<tr>
<td>6 Digit</td>
<td>16</td>
</tr>
<tr>
<td>7 Digit</td>
<td>15</td>
</tr>
<tr>
<td>8 Digit</td>
<td>14</td>
</tr>
<tr>
<td>9 Digit</td>
<td>12</td>
</tr>
</tbody>
</table>

- **Chapel**
- **Python**
Machine Learning

- Latent Dirichlet Allocation
- Data set?
  - Stored as doc/word count matrix
    - 6906 Words across 3000 Documents
- Performance for computing T topics
  - $T = \{ 2, 4, 8, 16, 32, 64 \}$
Machine Learning

Model Training

<table>
<thead>
<tr>
<th>Topics</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>21000</td>
</tr>
<tr>
<td>4</td>
<td>21000</td>
</tr>
<tr>
<td>8</td>
<td>21000</td>
</tr>
<tr>
<td>16</td>
<td>21000</td>
</tr>
<tr>
<td>32</td>
<td>21000</td>
</tr>
<tr>
<td>64</td>
<td>21000</td>
</tr>
</tbody>
</table>

- Chapel
- Octave
Machine Learning

References – Latent Dirichlet Allocation


Current Work
Current Projects

• Resilient Key-Value storage for Chapel
  – Google's Big Table

• Log-Structured Merge Tree
  – Append-only log
  – Transaction is a tree
  – Transaction buffer is a forest
  – Compact forest operation

• Distributed domains/dmap support

• Implementation in progress
Current Projects

• Directed Acyclic Graph processing for Chapel!
  – Tensorflow, Dask, Storm, Heron, Spark, Theano, etc

• Users build execution DAGs, runtime executes the DAG

• Graph optimizations/transformations
  – Optimization/Simplification/Computer Algebra (auto-differentiation)
  – Scheduling
  – Communications
  – Track Graph Execution for “replay/recovery”

• Prototype implementation – basic “calculator math”
  – Works for scalar-scalar and vector-vector
  – scalar-vector should be easy – has been problematic
Thank you!

- Chapel Team
- GASNET Team
- Questions?