Towards Large-Scale Matrix Completion Using XSEDE/Bridges

David Hughes
dmhughes@cpp.edu

Computational Intelligence Group
Computer Science Department
Cal Poly Pomona
About me

- Senior at Cal Poly Pomona
- I am currently working with Dr. Hao Ji on two research projects through the XSEDE EMPOWER program:
  - “Novel Randomized Algorithms for Large-Scale Matrix Completion”
  - “Parallel Implementation of Block Matrix Operations in Distributed Computing Systems”
## XSEDE

- **The Extreme Science and Engineering Discovery Environment**

- **XSEDE** is an NSF-funded virtual organization

<table>
<thead>
<tr>
<th>Center (Site)</th>
<th>Organization</th>
<th>Type</th>
<th>Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>IU/TACC (Jetstream)</td>
<td>Indiana U</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>IU/TACC Storage (Jetstream Storage)</td>
<td>Indiana U</td>
<td>storage</td>
<td>User Guide</td>
</tr>
<tr>
<td>LSU Cluster (superMIC)</td>
<td>LSU CCT</td>
<td>compute</td>
<td></td>
</tr>
<tr>
<td>Open Science Grid (OSG)</td>
<td>OSG</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>PSC Bridges GPU (Bridges GPU)</td>
<td>PSC</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>PSC Large Memory Nodes (Bridges Large)</td>
<td>PSC</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>PSC Regular Memory (Bridges)</td>
<td>PSC</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>PSC Storage (Bridges Pylon)</td>
<td>PSC</td>
<td>storage</td>
<td>User Guide</td>
</tr>
<tr>
<td>SDSC Comet GPU Nodes (Comet GPU)</td>
<td>SDSC</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>SDSC Dell Cluster with Intel Haswell Processors (Comet)</td>
<td>SDSC</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>SDSC Medium-term disk storage (Data Oasis)</td>
<td>SDSC</td>
<td>storage</td>
<td>User Guide</td>
</tr>
<tr>
<td>Stanford University GPU Cluster (XStream)</td>
<td>Stanford U</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>TACC Data Analytics System (Wrangler)</td>
<td>TACC</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>TACC Dell/Intel Knights Landing, Skylake System (Stampede2)</td>
<td>UT Austin</td>
<td>compute</td>
<td>User Guide</td>
</tr>
<tr>
<td>TACC Long-term Storage (Wrangler Storage)</td>
<td>TACC</td>
<td>storage</td>
<td>User Guide</td>
</tr>
<tr>
<td>TACC Long-term tape Archival Storage (Ranch)</td>
<td>TACC</td>
<td>storage</td>
<td>User Guide</td>
</tr>
</tbody>
</table>
Bridges

- Pittsburgh Supercomputing Center (PSC) Resource
- Virtual tour
  - [https://www.psc.edu/index.php/bridges-virtual-tour](https://www.psc.edu/index.php/bridges-virtual-tour)
- Bridges Resources
  - PSC Bridges GPU (Bridges GPU)
  - PSC Large Memory Nodes (Bridges Large)
  - PSC Regular Memory (Bridges)
    - Interact -N 4
  - PSC Storage (Bridges Pylon)
    - $SCRATCH
## Bridges Node Types

<table>
<thead>
<tr>
<th>Type</th>
<th>RAM</th>
<th>Phase</th>
<th>n</th>
<th>CPU / GPU / other</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESM</td>
<td>12TBb</td>
<td>1</td>
<td>2</td>
<td>16 × Intel Xeon E7-8880 v3 (18c, 2.3/3.1 GHz, 45MB LLC)</td>
<td>HPE Integrity</td>
</tr>
<tr>
<td></td>
<td>12TBc</td>
<td>2</td>
<td>2</td>
<td>16 × Intel Xeon E7-8880 v4 (22c, 2.2/3.3 GHz, 55MB LLC)</td>
<td>Superdome X</td>
</tr>
<tr>
<td>LSM</td>
<td>3TBb</td>
<td>1</td>
<td>8</td>
<td>4 × Intel Xeon E7-8860 v3 (16c, 2.2/3.2 GHz, 40 MB LLC)</td>
<td>HPE ProLiant DL580</td>
</tr>
<tr>
<td></td>
<td>3TBc</td>
<td>2</td>
<td>34</td>
<td>4 × Intel Xeon E7-8870 v4 (20c, 2.1/3.0 GHz, 50 MB LLC)</td>
<td></td>
</tr>
<tr>
<td>RSM</td>
<td>128GBb</td>
<td>1</td>
<td>752</td>
<td>2 × Intel Xeon E5-2695 v3 (14c, 2.3/3.3 GHz, 35MB LLC)</td>
<td>HPE Apollo 2000</td>
</tr>
<tr>
<td>RSM-GPU</td>
<td>128GBb</td>
<td>1</td>
<td>16</td>
<td>2 × Intel Xeon E5-2695 v3 + 2 × NVIDIA Tesla K80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>128GBc</td>
<td>2</td>
<td>32</td>
<td>2 × Intel Xeon E5-2683 v4 (16c, 2.1/3.0 GHz, 40MB LLC) + 2 × NVIDIA Tesla P100</td>
<td></td>
</tr>
<tr>
<td>DB-s</td>
<td>128GBb</td>
<td>1</td>
<td>6</td>
<td>2 × Intel Xeon E5-2695 v3 + SSD</td>
<td>HPE ProLiant DL360</td>
</tr>
<tr>
<td>DB-h</td>
<td>128GBb</td>
<td>1</td>
<td>6</td>
<td>2 × Intel Xeon E5-2695 v3 + HDDs</td>
<td>HPE ProLiant DL380</td>
</tr>
<tr>
<td>Web</td>
<td>128GBb</td>
<td>1</td>
<td>6</td>
<td>2 × Intel Xeon E5-2695 v3</td>
<td>HPE ProLiant DL360</td>
</tr>
<tr>
<td>Othera</td>
<td>128GBb</td>
<td>1</td>
<td>16</td>
<td>2 × Intel Xeon E5-2695 v3</td>
<td>HPE ProLiant DL360, HPE ProLiant DL380</td>
</tr>
<tr>
<td>Gateway</td>
<td>64GBb</td>
<td>1</td>
<td>4</td>
<td>2 × Intel Xeon E5-2683 v3 (14c, 2.0/3.0 GHz, 35MB LLC)</td>
<td>HPE ProLiant DL380</td>
</tr>
<tr>
<td></td>
<td>64GBc</td>
<td>2</td>
<td>4</td>
<td>2 × Intel Xeon E5-2683 v3</td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td>128GBb</td>
<td>1</td>
<td>5</td>
<td>2 × Intel Xeon E5-2680 v3 (12c, 2.5/3.3 GHz, 30 MB LLC)</td>
<td>Supermicro X10DRi</td>
</tr>
<tr>
<td></td>
<td>256GBc</td>
<td>2</td>
<td>15</td>
<td>2 × Intel Xeon E5-2680 v4 (14c, 2.4/3.3 GHz, 35 MB LLC)</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>281.75TB</strong></td>
<td><strong>908</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

John Urbanic 2018
Modules

- Environment management package for using certain software
- Command: `>module avail software-name`

Example Usage: (after interact)

`>module load hadoop`
`>start-hadoop.sh`

https://www.psc.edu/user-resources/software/module
https://www.psc.edu/resources/software
Hadoop and Spark on Bridges

- **Spark**
  - HDFS as data source
  - Yarn as resource management and job scheduling/monitoring

- **HDFS**
  - Uses $SCRATCH disk space (pylon5)
  - 1. Transfer files using scp/sftp
  - 2. Use command: `>hdfs dfs -put`

- **YARN**
  - Retrieve full output (primarily error logs)
  - `>yarn logs -applicationId “applicationid”`

https://www.psc.edu/bridges/user-guide/hadoop-and-spark
More about Hadoop and Spark
Hadoop

History

- Open source project called Hadoop created by Doug Cutting in 2005
- Based on the MapReduce paper by Google
- Now Apache Hadoop

What it does

- Runs using the MapReduce algorithm
- Data is processed on different nodes of the cluster system
- Solution for storing and processing voluminous unstructured data sets
Hadoop Framework

- Hadoop Common
  - Java libraries and utilities needed to start Hadoop
- Hadoop YARN
  - Job Scheduling and cluster resource management
- Hadoop Distributed File System (HDFS)
  - Distributed file system for access to application data
- Hadoop MapReduce
  - Parallel processing of large data sets
Apache Spark
Spark

- Research project at UC Berkeley AMPLab
- “Spark: Cluster Computing with Working Sets,” 2010
- Goal: “Support a much wider class of applications than MapReduce, while maintaining automatic fault tolerance” [2]
Goal: “Support a much wider class of applications” [2]

- Multipass applications with low-latency data sharing requirements
- Common in data analytics
  - Iterative algorithms: machine learning and graph algorithms
  - Interactive data mining: load data into RAM across a cluster and query repeatedly
  - Streaming Applications: maintain aggregate state over time
Goal: unified engine across data sources, workloads and environments

Source: Databricks
Why is MapReduce and variants not efficient?

- “These systems are built around an acyclic data flow model” [1]
- The application runs a “series of distinct jobs, each of which reads data from stable storage and writes back to stable storage” [2]
- “Significant cost loading the data on each step and writing it back” [2]
Solution to Too Much Data Movement

- Resilient Distributed Datasets (RDDs)
- “RDDs can be stored in memory between queries without requiring replication” [2]
- “Fault tolerant, parallel data structures that let users” [6]
  - “Explicitly persist intermediate results in memory”
  - “Control their partitioning to optimize data placement”
  - “Manipulate them using a rich set of operators”

Thus, the main feature of Spark is its in-memory cluster computing
RDD Dataflow

Based on Source Graphic by Earl Lawrence
Generality

- Spark was built with the idea of supporting a wider class of applications

- Java, Scala, Python, R, and SQL
MLlib

- Spark’s open-source distributed machine learning library [9]
- “Targets large-scale learning settings that benefit from data-parallelism or model-parallelism”
- “Fast and scalable implementations of standard learning algorithms for common learning settings”
  - Includes classification, regression, collaborative filtering, clustering, and dimensionality reduction
GraphX

- “Distributed graph computation framework that unifies graph-parallel and data-parallel computation” [10]
  - Graph-parallel: partitions graph data across nodes then resolves dependencies through iterative computation and communication
- **Why?** Previous systems restricted the allowed computations to increase efficiency, but as a result they have to use external systems to compose the graph-parallel and data-parallel computations
  - “Extensive data movement and complicated programming model”
- Operators: subgraph, joinVertices, mapReduceTriplets etc.
- Included Algorithms: PageRank, Connected Components, Triangle Counting

Using Spark
Submitting Spark Jobs

- Scala: package code into jar using sbt or maven
- Python: “--py-files” argument to add python files

Example:
spark-submit --class Main --master yarn --deploy-mode cluster --driver-memory 115g --executor-memory 115g - -conf spark.driver.maxResultSize="0" rsvdtest.jar mtxr matrix_10k.mtx 10 $HOME//rsvdtest/output_10k.txt 1 3

More: https://spark.apache.org/docs/latest/submitting-applications.html
SBT

Scala Interactive Build Tool

Commands:
- sbt ~run
- sbt package

Directory Structure:
```
example-project
  |--build.sbt
  |--main
    |--scala
      |--Main.scala
```

```
build.sbt

name := "Example-Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += groupId %%% artifactId % revision

Example library dependency:
libraryDependencies += "org.apache.spark" %%% "spark-core" % "2.3.0"
libraryDependencies += "org.apache.spark" %%% "spark-graphx" % "2.3.0"
libraryDependencies += "org.apache.spark" %%% "spark-mllib" % "2.3.0"
```

https://www.scala-sbt.org/index.html
Using Shell

> pyspark

> spark-shell

These already have SparkContext created as “sc”
Programs will require creating sc in the file

Python

from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("Test_App")
sc = SparkContext(conf = conf)

Spark

import org.apache.spark.SparkContext._
import org.apache.spark._
val conf = new SparkConf()
    .setAppName("RandomSVD")
val sc = new SparkContext(conf)
Standalone mode

- Good for debugging and running jobs locally before interacting/batch with Bridges

Example command:
spark-submit --class Main --master local target/scala-2.11/rsvdtest_2.11-1.0.jar mtxr paperP.mtx 3 output.txt 1 3
Installing PySpark on Windows

- Install Java JDK
  - Add JAVA_HOME environmental variable
- Install Python 3
  - Select option to add Python to PATH
- Download a Hadoop Binary (Windows specific issue)
  - Recall that Spark uses HDFS to work with files rather than NTFS like Windows
  - Extract to any directory ..\Hadoop
  - Create a HADOOP_HOME environment variable pointing to the installation folder
    - Add %HADOOP_HOME%\bin to the Windows Path variable
- In command line: pip install pyspark
Scala Spark

Installed similarly, differences include

- Download a prebuilt Spark version
- Extract to a Spark folder
- Create a SPARK_HOME variable
- Add `%SPARK_HOME%\bin` to PATH

Spark-shell and spark-submit will now work on command prompt

Full list of instructions: [https://medium.com/@josemarcialportilla/installing-scala-and-spark-on-windows-249632e6b83b](https://medium.com/@josemarcialportilla/installing-scala-and-spark-on-windows-249632e6b83b)
Examples
Example: Simple Counting

rdd = sc.textFile("shakespeare.txt")

rdd.count()  # number of lines

words = rdd.flatMap(lambda x: x.split() )

words.count()  # number of words

words.distinct().count()  # number of distinct words

word_counts = words.map(lambda x: (x,1))  # create an rdd of tuples

word_counts.reduceByKey(lambda x,y: x+y)  # reduce tuples by word

result = word_counts.map(lambda x: (x[1], x[0]))  # swap key and value

result.sortByKey(False)  # sort by key, descending

result.take(5)  # collect the top 5 (count, word) pairs
Our research projects:
• “Novel Randomized Algorithms for Large-Scale Matrix Completion”
• “Parallel Implementation of Block Matrix Operations in Distributed Computing Systems”
Novel Randomized Algorithms for Large-Scale Matrix Completion

- Idea is to implement techniques in new ways using primarily Scala and Apache Spark GraphX
  - Verify the correctness of the solution (usually locally)
  - Benchmark and make comparisons made using the Bridges HPC resource on large datasets

- Datasets that I commonly use:
  - **webbase-1M**: 1,000,005 by 1,000,005 matrix ; 3,105,536 nonzeros ; 69.0 MB
  - **Cage15**: 5,154,859 by 5,154,859 matrix ; 99,199,551 nonzeros ; 2.57 GB

Suite Sparse Matrix Collection: [https://sparse.tamu.edu/](https://sparse.tamu.edu/)  
[https://sparse.tamu.edu/Williams/webbase-1M](https://sparse.tamu.edu/Williams/webbase-1M)  
[https://www.cise.ufl.edu/research/sparse/matrices/vanHeukelum/cage15.html](https://www.cise.ufl.edu/research/sparse/matrices/vanHeukelum/cage15.html)
The objective of **matrix completion** is to recover the missing (unknown) entries of an incomplete matrix from a small subset of observed ones.
Predicting As Matrix Completing

- The objective of matrix completion is to recover the missing (unknown) entries of an incomplete matrix from a small subset of observed ones.
Singular Value Decomposition

- A factorization of a real matrix $A \in \mathbb{R}^{m \times n}$ is singular value decomposition (SVD) if

$$A = U \times \Sigma \times V^T$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose elements $\sigma_1, \sigma_2, \ldots, \sigma_n$, are nonnegative singular values in non-decreasing order.
Singular Value Decomposition

- A factorization of a real matrix $A \in \mathbb{R}^{m \times n}$ is singular value decomposition (SVD) if
  \[ A = U \times \Sigma \times V^T \]
  where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose elements $\sigma_1, \sigma_2, \ldots, \sigma_n$, are nonnegative singular values in non-decreasing order.

low rank approximation $A_k$
Singular Value Decomposition

- A factorization of a real matrix $A \in \mathbb{R}^{m \times n}$ is singular value decomposition (SVD) if
  \[ A = U \times \Sigma \times V^T \]
  where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose elements $\sigma_1, \sigma_2, \ldots, \sigma_n$, are nonnegative singular values in non-decreasing order.

The traditional deterministic SVD algorithm,
- takes time in $O\left(\min(mn^2, nm^2)\right)$
- is unsalable for applications involving large scale dataset.

low rank approximation $A_k$
Introducing Randomness

“Randomized Matrix Decompositions in R”
arXiv:1608.02148v4
Randomized SVD

- "Randomized Matrix Decompositions in R"

- Facebook research also performs the same method
Randomized SVD

- Randomized Matrix Decompositions
- Facebook research also performs the same method

Steps involved:
- Generation of random matrix $\Omega$;
- Matrix-matrix multiplication of $A\Omega$ to produce $Y$;
- QR decomposition on $Y$;
- Matrix-matrix multiplication of $Q^TA$;
- Deterministic SVD decomposition on $B$.

arXiv:1608.02148v4
Representation of the Sparse Matrix and Block

- Property Graph in Spark’s GraphX module
  - represent a large, sparse matrix in a distributed manner
  - is built on the top of Spark’s Resilient Distributed Dataset (RDD) abstraction, which has

```scala
class Graph[ VertexType, EdgeType ] {
  val vertices: RDD[ (VertexId, VertexType) ]
  val edges: RDD[ Edge(SrcId, DstId, EdgeType) ]
}
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Representation of the Sparse Matrix and Block - Cont.

- To represent the sparse matrix and the block together

```scala
val g: Graph[Array[Double], Double].
```

Vertex type  Edge type
Representation of the Sparse Matrix and Block - Cont.

- To represent the sparse matrix and the block together

```
val g: Graph[Array[Double], Double].
```

$$P = \begin{bmatrix}
0 & 0.0002 & 0 & 0.6690 & 0.9989 \\
0 & 0 & 0.9998 & 0 & 0.0011 \\
0 & 0.9998 & 0 & 0 & 0 \\
0.9090 & 0 & 0.0002 & 0 & 0 \\
0.0910 & 0 & 0 & 0.3310 & 0 \\
\end{bmatrix}$$

$$X_0 = \begin{bmatrix}
0.0925 & -0.1550 & -0.2292 \\
0.3155 & -0.0514 & 0.5154 \\
-0.3886 & 0.0406 & 0.1232 \\
0.1483 & 0.4243 & -0.0107 \\
0.0548 & 0.3284 & 0.1213 \\
\end{bmatrix}$$
Sparse Matrix-Block Multiplication

- To generate the next block $X_i$ from $X_{i-1}$, we need the sparse matrix-block matrix multiplication

\[ X_i = P X_{i-1} \]
Sparse Matrix-Block Multiplication

● To generate the next block $X_i$ from $X_{i-1}$, we need the **sparse matrix-block** matrix multiplication

$$X_i = PX_{i-1}$$

● In fact, each row in $X_i$ can be obtained from

$$X_i^{(j)} = \sum_{k \in \Omega_j} p_{jk}X_{i-1}^{(k)}$$

where $X_i^{(j)}$ denotes the $j$th row vector in $X_i$

To take advantage of graph-parallel computation in Spark, we use aggregation operation on the property graph
Scala Code for SparseMatrix-Block Multiplication

//Y = P' * Q
   Yi = gi.aggregateMessages[Array[Double]] (  
      sendMsg = ctx =>
      ctx.sendToDst(ctx.srcAttr.map(_*ctx.attr)),
      mergeMsg = _.zip(_).map{case(x,y)=> x+y},
      new TripletFields( true , false , true )
   )

//Y = P * Y
   Yi = gi.aggregateMessages[Array[Double]] (  
      sendMsg = ctx =>
      ctx.sendToSrc(ctx.dstAttr.map(_*ctx.attr)),
      mergeMsg = _.zip(_).map{case(x,y)=> x+y},
      new TripletFields( false , true , true )
   )
More functions

\[
\begin{align*}
Z_i &= (P - I)Q_i \quad \text{// Sparse matrix-block multiplication} \\
B_i &= Z_i^T Z_i \quad \text{// Dense matrix multiplication} \\
V_i \Lambda_i V_i^{-1} &= B_i \quad \text{// Eigendecomposition} \\
Y_i &= Q_i V_i \quad \text{// Dense matrix multiplication}
\end{align*}
\]
References


Questions/Comments?
Thank You