Dylan Hutchison, dhutchis@stevens.edu, 862-226-2764, www.cs.stevens.edu/~dhutchis
Github denine99

30 August 2014
Other projects we should compare to: GraphLab, C++ distributed machine learning at scale, developed 4 years at Carnegie Mellon

1 September

- Possibility to turn this into a product in the same manner as Sqrrl. We could create a library that targets heterogeneous machine architectures (GPUs and FPGAs). It will use open-source libraries like Accumulo and the graph library coming out of CSAIL, but there is no outright requirement that our library be open-source.
- Accessed Ganesan's server. It doesn't have Git ¬.¬ Not sure whether we need Sudo access.
    - See Ganesan server doc
    - What experiments shall we run first? Perhaps install Accumulo, but we don't need it for initial graph algorithm work. Maybe just get an adjacency matrix in a text file and go from there.
    - Todo: look back at old CS 182 graph code. (edit:done.Nothing of much interest)
- Researched platforms to host code. Github organizations are free if we go open source. Otherwise, let's use an Atlassian Bitbucket team, free for unlimited users with private repos since we all qualify for an academic license.
- Found an excellent overview of Accumulo from Hortonworks: http://hortonworks.com/hadoop/accumulo/

2 September - class Tuesday
Need to focus the project away from research. Use graph algorithms that already exist. Implement them traditionally, then implement targeting Ganesan's cluster- GPUs and FPGAs. Get a deliverable library.
I advise taking Accumulo, Hadoop, Zookeeper, D4M, etc. for granted. Use them, so we don't have to worry about their implementations. Later if we get to an advanced enough state, we could consider it.

Also relax FPGAs. They are cool, but we should focus on one architecture at a time.

3 September
Link Prediction paper

- Jaccard's coefficient: $\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$ where gamma means the neighbors of the node measure of similarity between nodes. Can be used in link prediction; see paper

4 September TG class
Unique value proposition: benefit to early adopter. (No features or appeal to mass market.)
Solution: top feature for each problem. Gauge: don't run overboard with features, hike still solving all problems. Features should be 1-1 with problems.
Channel: how do early adopters find your product? US: publish performance numbers in industry journals, direct market to companies, word of mouth in our specialized community.
Cost: all in labor at first, unless we need licenses for Matlab or a GPU toolkit or something.
Metrics: Performance! Flops, scale wth number of nodes, number of cores, number of GPUs, etc.
Unfair Advantage: Targeting GPUs and FPGAs. We built the algorithms no one else has; takes a lot of study and time.

Team Meeting --
Present: Di, Eric, Yao, Xuelian, Dr. Ganesan
We still have a people problem: we have two ECE members: Eric and I. Perhaps we can convince Dr. McNair to let Di take the place of a third ECE member, since his grade is totally dependent on Dr. Ganesan by the permission of Dr. Klappohltz, head of CS senior design.
We will try and recruit another member.

Possible architecture libraries: CUDA, OpenCL, Xilibus driver for FPGAs
Possible languages: Matlab, Java, Python, C++

We want to choose a language with the easiest GPU support, since we can implement graph algorithms in any language. Higher-level is better. Maybe reimplement in C later if we truly want line performance. But as Dr. Ganesan says, this may be only a constant factor better.
My todo: research the Matlab graphics toolkit; see if we have a Stevens license
Possible algorithms: shortest paths, strongly connected components, triangle counting, …
Possible datasets: best case is to use the *same dataset as another performance paper*. Then we can run our algorithms on the same dataset and directly compare.

Graph types: Erdos-Renyi and Power Law

The team generally agrees that bringing databases into the picture will make our work really make a difference. Few research efforts create an entire pipeline; they just build a proof of concept tool that no one uses. But we need to focus on the basics first. Implement a graph algorithm with no frills attached, then think about how we can integrate with databases later.

5 September

Not sure if we have access to the Matlab Parallel Computing Toolbox
Well-explained webinar video here on GPU computing in Matlab
Use 'ver' to see what toolboxes are installed.
Jaroor continues to experiment with D4M; met for an hour.
NOPE, we do not have a Stevens site license to Matlab Parallel Computing Toolbox
Conference call with MIT folks
To lookup: **Accumulo iterator power**. Refresh SVDs and QC decomposition for non-neg matrix factorization.

Stack Overflow question on Java and CUDA
1 GPU is best for *Data Parallel*, not Task Parallel
2 Also best for *compute bound*, not memory bound.
3 Data locality - GPUs have a small, fast-access cache memory. Try to stay within cache.
Challenges:? Occupancy, register pressure, shared memory pressure, memory coalescing
"broader support for OpenCL in the Java world" -- OpenCL is vendor-independent. Use case for CUDA is if we want the special BLAS libraries there.

https://code.google.com/p/aparapi/
Looks good. I like the abstraction. It is recent (last commit May 2014) and targets future functional programming with Java 8 lambdas (optional, don't be scared) and the HSA language. Automatically runs in Java threads if a GPU is not available.

Downloaded binaries for Zookeeper, Hadoop, Accumulo, tried to run but failed. Don't try to run Accumulo on Windows… cygwin is a pain.
Got source for Accumulo. Installed Eclipse. Had to install 64-bit Java to match 64-bit Eclipse.
Eclipse configuration tips at the bottom here https://accumulo.apache.org/source.html
Installed M2E Maven for Eclipse. Eclipse cheatsheet.

6 September
We have a team! Created Github team Stevens-GraphTeam on Stevens-GraphGroup.

As before, the key is to get started. Here is a list of some things we will need to work on, now or later:

1. Fix a language. For now I assume **Java**.
    a. (It's okay if you want to use C++; you can implement in C++ if you are more comfortable playing in C++. But we should fix a main language for the team.)
    b. By the way, this C library may be helpful: **Combinatorial BLAS** Library (related to GraphBLAS)
    The MIT Team may not use it directly but will draw inspiration on the basic "graph building blocks" from it.
2. Experiment with different **data structures**. To start, naive two-dimensional arrays of doubles are fine.
3. Implement graph algorithms. Here's a few to start:

    a. Shortest Paths
    b. Strongly Connected Components
    c. Triangle Counting
4. Figure out a good way (using some library?) to **benchmark** performance.
    a. Imagine a method that takes a Runnable object, and times how long it takes to run it. Again, this is a starting point for design. We might imagine that we want more fine-tuned algorithm timings, e.g. timing how long the first part of an algorithm takes, then the second. Or the communication time vs. computation time, etc.
5. Experiment with OpenCL bindings (or CUDA; see the comparison for why OpenCL may be a better choice than CUDA unless we can use the BLAS libraries of CUDA)
6. Explore **related work**. Compare to our work, show how the related work is similar/different, etc.
7. Put together a website for our project. We will start with basic things like a front page project description, a page describing our team members (all of us need to write a one paragraph bio at some point), a page describing our results with images and text and graphs.
    a. If we can get there, an awesome next step is adding interactivity. We can do this by
        i. User interactivity, via user requests by AJAX
        ii. Server interactivity, actually connecting Ganesan's cluster to the website to handle incoming requests (maybe to execute some analytic on a preloaded dataset)
8. Think of how we will visualize our performance. It is ideal if we can generate graphs from our Java code.
9. Do misc paperwork. For the senior design team--Eric, Xin, Hefei and Dylan-- we must submit two reports every week:
    a. Status Report on the whole group (submit 1/week for the whole group)
    b. Effectiveness Survey (each of us submits 1/week)
10. We also have paperwork/labwork for TG 421.

D4M is two things:
(1) A set of software for doing analytics.
(2) A schema for ingesting and indexing diverse data into a NoSQL database like Accumulo

Eclipse with m2e plugin to do maven projects works! I added

```
<dependency>
    <groupId>org.apache.accumulo</groupId>
    <artifactId>accumulo-core</artifactId>
    <version>1.6.0</version>
</dependency>
```

to my pom.xml of a test Maven project and it worked-- it downloaded ALL the JARs, with sources and javadoc. Yay-- I navigated dependency hell. Use the Eclipse Maven GUIs.

8 September

Hi Eric,

Yes, it's a good book but easy to get overwhelmed when flipping through. **Read smartly**. I advise reading with pencil and paper at your side, so that you can try things on sample graphs.

Chapter 1 is intro material and chapter 2 is on general principles of linear algebra. They are both nice to know but not strictly necessary. Skip them if you want to go directly to your first graph algorithm.

Chapter 3 and 4.3 are tutorials on first graph algorithms. You will learn about (strongly) connected components and shortest paths in two different ways, and some other related constructs. These are good places to start.

Chapter 5 starts getting serious with Bellman-Ford shortest paths, all-pairs shortest paths and minimum spanning tree.

Future chapter I haven't looked at much yet. We will not use all the material in them, but probably some.

When you read the stuff about semirings, the book is referring to replacing the '+' and 'x' operations of matrix multiplication with something else, such as 'min' and '+'.

~Dylan

9 September 2014

Idea: write native C or C++ code for GPU programming.
Then use JNI on an Accumulo iterator to call the C code for GPU use with OpenCL.
Could use Julia to write the GPU graph handling compute code. Then compile Julia to a C executable.
Scans in parallel are fine. Use the same scanner object-- lock it and get an iterable, then unlock it for next thread (so just synchronize the scanner).

Jeremy: Accumulo community deprecated the connector authentication structure that LLgrid uses. Need a complex rewrite of the authentication to match the new API.
University cluster administration has gone backward. Most university clusters are ‘piles’, not clusters. Just a collection of nodes with Hadoop and some other stack stuff that is hard to support. Hacking those clusters to make them work is kind of a waste of time.
TX-E1 has 10 nodes with GPUs-- much more likely to work.

Wade Vinson, HP Data Centers - super modular and customizable. Put Data center trailors around the world.

What features are in BigTable/HBase but not Accumulo?
Andrew Wells, Ilya Dynin - Accumulo team at Clearedge IT - awesome folk, have their own private library on top of Accumulo

> Hi folks,
> This stack overflow answer contains a treasure trove of links, advice and info on GPU computing in Java.
> http://stackoverflow.com/questions/22866901/using-java-with-nvidia-gpus-cuda
>
> The author makes a convincing argument for **OpenCL** over CUDA, because OpenCL is an industry standard / not vendor-specific. Unless we want to use the BLAS routines of CUDA that is.
>
> So far my favorite is the **Aparapi** library, but I to be fair I did not look at all of them. Java is a good candidate.
>
> Some random notes:
> 1 GPU is best for *Data Parallel*, not Task Parallel
> 2 Also best for *compute bound*, not memory bound.
> 3 Data locality - GPUs have a small, fast-access cache memory. Try to stay within cache.
>
> Cheers,
> Dylan
>
> Hi, Dylan,
> I have read the document you gave to me. I agree the idea that the OpenCL is a better choice than the CUDA, because although the latter one is supported by the Nvidia and has a good developing toolkit, but the OpenCL is more popular in the industry and academic. since our purpose is to design an algorithm that can be widely used, the OpenCL is better.
>
> However, I think the java is not a good choice to do the job. The OpenCL is based on the C, and the java-based GPU computing is not a mature area. We need to use the JNI that enables JAVA code running in a JVM to call and be called by native applications and libraries written in other languages such as C and C++. The pitfall is that there will be some bugs needs to be fix and it also influence the performance, which is essential for an algorithm.
>
> withOpenCL and CUDA, considering that we need to do a lot of work and spend a considerable of times on fixing the bug and the performance , I believe the c++ or c, which is more bottom layer language and more stabilize will be our better choice.

Regards,
Mojie Yao

Yao, thank you for your analysis. You have coincidental timing-- I started thinking along the same lines over the course of today at the conference.

All, Yao made a good argument for adopting a C-like language instead of Java-- that the Java OpenCL libraries are not mature and may introduce bugs or inexplicable performance hits from within the Java OpenCL libraries. I noticed that most of them just rewrite calls to C/C++ anyway, making it an attractive option as long as you're comfortable writing in C++. Better to write something in some language than nothing in C++.

If you do write in a language other than C/C++, just make sure it has a compilation to OpenCL C/C++ that we can drop in. We can also rewrite it in native C/C++ if we have to for performance reasons.

On the note of Accumulo which has bindings written in Java, we can call the C/C++ native OpenCL code, copied to each node of the Accumulo instance, using JNI. I talked to professionals from Clearedge IT today and they agreed this is the most reasonable approach. To reiterate, we

1. Get a C/C++ OpenCL binary for doing computations on a GPU. This is either from writing in C/C++ natively or from writing in another language that compiles down to the C/C++ OpenCL code. I believe any language with OpenCL support can do this, Java, Julia, etc.
2.
    1. Yao makes a good point that writing natively in C/C++ is safest. But don't be afraid to write in another language if it is easier to get started in that or easier to write high level code.
3. When we get to integrating with Accumulo, we will distribute our C/C++ OpenCL binary to every node in Accumulo. Then call that code from Java using JNI.

If you are confused, the analysis boils down to this:
Write in C/C++ if comfortable. If uncomfortable or you see an easy way to write something in a higher level language, then go ahead in write in that language. We can always translate your code when we start worrying about performance. Get the algorithms right first.

How's that sound? Everyone on the same page? Feel free to discuss with each other or me or on the Thursday 2:00pm time slot if you wish.

Regards,
Dylan Hutchison

I foresee two styles of operation in Accumulo. I illustrate with the task of maintaining a degree distribution. Assume a (naive) schema of one table where the rows are nodeIDs and the columns are the nodes that that node connects to (out-edge table) AND another table where the rows are nodeIDs and the columns are the nodes that connect to that node (in-edge table). (maybe a third table lists all the edges). We're interested in the in- and out- degree distribution.

1. Extra insert mode.
    a. Maintain extra table in-degree and out-degree, both with summing combiners.

   b. On addition of a new edge A -> B,
      i. add (A,B,1) to the out-edge table
      ii. (A,out-deg,+1) to the out-edge-deg table
      iii. (B,A,1) to the in-edge table
      iv. (B,in-deg,+1) to the in-edge-deg table. Each edge addition is 4 writes.
   c. Obtain the in- and out- degree distributions in one of two ways
      i. (dumb way) query down the whole out-edge-deg and in-edge-deg tables and make a histogram of the degrees of the nodes
      ii. (smart iterator way) use a scan-time iterator when querying the out- and in-edge-deg tables that does the equivalent of a StatsCombiner. The iterator incrementally forms the degree distribution by retaining the number of counts of each degree #, e.g.,
         1. start with empty sparse vector,
         2. then when passed a row with a node with degree 7 from next(), vector is [7=1],
         3. then when passed degree 3, vector is [3=1, 7=1],
         4. then when passed degree 7, vector is [3=1, 7=2],
         5. … one row is returned by the query -- the final degree dist. vector
2. MapReduce mode. Don't use extra tables. Instead, periodically (or continually) initiate a MapReduce job with AccumuloInputFormat and reduce the table to the degree distribution.
3. (compromise) Don't use extra tables. Instead use a wholeRowIterator that counts the number of columns in the row and just returns the number of columns. Chain that into a second iterator like #1 that forms the degree distribution. #1 retains the degrees at ingest time while this approach computes the degrees on the fly at the query time. I think this is ok even in the face of supernodes with tons of connections, because this happens at the tablet server before results are sent from the tablet server to the client calling code.
4. Mark all the "accounted for" triples added with a special designation. Those without the special designation are marked and then accounted for in the degree tables at compaction time (maybe scan time too?). Or map reduce job??

Community similarity: how similar are graph communities - Jaccard coefficient
⇒ similar to intersecting iterator, and we can modify it to make a unifying iterator for the denominator 'or' component

MapD -- database on GPUs
https://txe1-portal.mit.edu/

Tons of resources here http://www.startup.ml/resources/

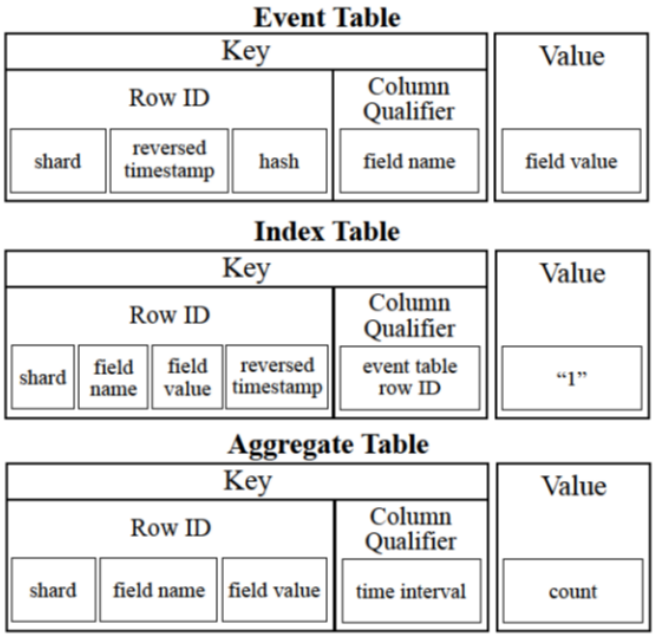
From Huy Nguyen at MIT LL: **FPGA**s are placed on flash memory. When wired together, you can refer to any of the memory accessible by an FPGA on the same address space. Great

possibility for iterator computation at the site of flash memory. Huy gave an argument for cost savings as compared to buying more memory too (but can't remember). Huy works on the BlueDBM FPGA-accelerated project.

Next meeting notes: related projects, FPGAs, OpenCL 1.2 or 2.0?

Ask questions: why do I study HPC and graphs and probability distributions? Why not math, or ….

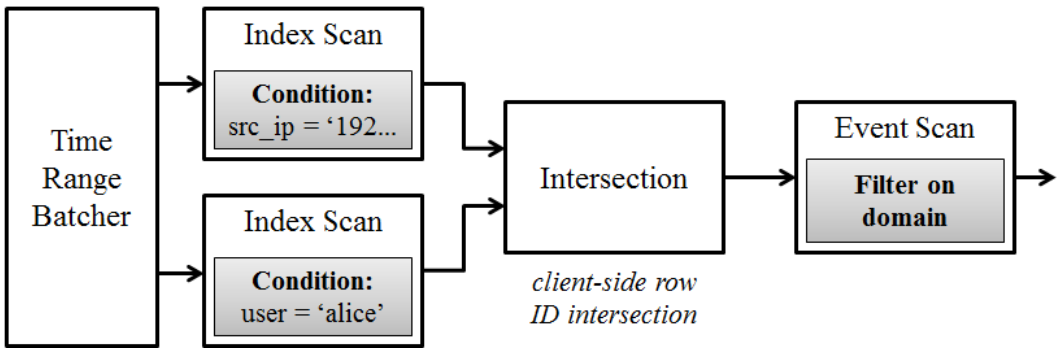
12 September
David's Cybersecurity group schema




```
WHERE (t BETWEEN t1 AND t2)
AND src_ip='192.168.0.1' AND user='alice'
AND domain='bigsite.com'
```

very selective

very common



Use the aggregate table to get the degrees for query optimization -- discover that the domain has huge degree, src_ip and user have small degree.
Goal: only want the hashes that meet all the criteria: timestamp between t1 and t2; src_ip matches; user matches.
Mode1: go directly to the event table. Use a filter iterator that excludes all the rows that do not meet all the criteria. Scans over all rows at the server side.
Mode2: go to the index table. Get the RowIDs of the rows that match the most selective criteria. Then goto event table, scan over the set of selected RowIDs, and choose the ones that match the common criteria with a server-side iterator.
Always filter time.
Why do intersection at the client? Why not do intersection on a node at the site of the database? Or even better do something like the intersecting iterator does. (idea0)
**Idea3: distribute obtained row IDs to different tablet servers based on predefined, known split points

Problem with intersecting iterator: only doable for within the same row.

Idea: client sends query to a node on the Accumulo database. The node does all the relevant scans and intersections and sends back final results over a stream to the client.
Idea2: do first scan on index table for condition with smallest degree; get back RowSetA. Pass RowSetA to an iterator that only returns rows within RowSetA to get RowSetB.

14 September
Implemented SCC in two ways, naive and using matrix inversion, for both dense and sparse matrices, and put it on Github. Some handy graph visualization there too.

There's something fishy about SYCL. I'm not sure what is up with it. There are tutorials here and here, so there some implementation of it?

As for the C++ wrapper, I see a version 2.0 but I'm not sure if the C++ wrapper made it there. Seems like we need input from Dr. Ganesan. The API looks kinda scary.

TinkerPop3 -- a cute, interesting stack for graph processing. I wonder how well it performs?
Distinguishes between OLAP global big processing that can transform the whole graph and OLTP local small transactional processing that transforms a local set of nodes in real time.
GraphML file format
Bulk Synchronous Parallel BSP style - uses barriers.
"Giraph makes use of the distributed graph computing paradigm made popular by Google's **Pregel**."
Implementation of TinkerPop spec in Accumulo: **AccumuloGraph**
Solution to Supernode blog post?
Article on graph computing to read.
There is clearly an industry community here; should learn about them.

Dr. Ganesan gave me sudo access. Set some things up.

16 September
Utility Patent stuff
Useful: run on graph data sets
Novel: implemented on heterogeneous architecture
Non-obvious part of our library: how we implement our graph algorithms.

Patent for 20 years. Large upfront cost. Stevens will pay the upfront cost in exchange for half of royalties.
Specification should have no undue experimentation. Claims define boundary.
Copyright is for expression of original work. Copyright software prevents people from word-for-word copying. People can take it, change variable names, and it is ok. Seems only useful if not open source.
Trademark is to protect a word / phrase.
Trade secret published.
Contrast all this with the open source world. Info is out there, we provide a service to support.
Publication is giving idea to someone not a co-inventor and not under non-disclosure.
Let's open-source. Don't worry about making money.

Xin - ECE tiger website

Notes for TG Lab 1

Target Customers: companies that have large data sets and want insight through top-performant, maybe even real time analytics. We specifically target analytics in the form of graph algorithms (fairly universal). Specifically, we target the CTO of companies with big data sets.

Problem	Solution
Need better graph algorithm performance for analytics	Use our library.
Want to use GPUs and FPGAs to improve performance, but we don’t know how.	Use our expertise and support. Pay us to help you figure out which analytics can speedup on GPUs and FPGAs, and help you use the library to realize those speedups.

Unique value proposition: better data analytic performance using heterogeneous architectures.

Channels: publish performance numbers in industry journals, direct market to companies, word of mouth in our specialized community.

Costs: all in labor / salaries. We need development time. Maybe some costs for software licenses or hardware.

Revenue: consulting fees, support agreements. Since we aim to open-source our library, primary revenue is by companies paying for a support agreement for us to help them setup, use and maintain our library.

Key metrics: flops, performance on benchmarks, scalability curves (computation time vs. number of GPUs, CPU cores, nodes, …)

Unfair Advantage: Targeting GPUs and FPGAs. We built the algorithms no one else has; takes a lot of study and time.

CPE 517 Computer and Digital Systems Architecture
EE 693 Heterogeneous Computing Architecture and Hardware

OpenCL code is complicated in its bare form. Too many calls, error codes, boiler plate context setup, etc. to worry about = too big a chance for error. We don't have time to become machine code gurus in a few months. I favor using a higher level library, even if we sacrifice some performance, the savings in frustration and learning time are well worth it. Plus, I bet the gain of going to barebones assembly is a constant factor.

17 September
Preparing Meeting Agenda for 18 September

- Review high level objective, project plan
- Complete team roles
- Ensure everyone is set with team logistics:
    - Weekly Report
    - Drive, Todoist, Github
    - Who needs help learning Git? Short tutorial on git after the meeting.
    - Journaling
    - Related Work
- Overview of FPGAs by Dr. Ganesan
- Walkthrough of SCCs on Matlab as a first simple algorithm

[All] Write a one paragraph bio and put in your journal

18 September - TG Class notes
Target customers: government, finance, social service
All would like real time solutions.

- Government: cybersecurity / national security monitoring, privacy,

- Finance: stock price prediction / valuation, risk analysis, simulations for different market conditions
- Social service: recommendations (you should friend this guy, like this page, etc. because they have good ties with you), advertisements

Stakeholders: Stevens Institute of Technology (we represent Stevens), MIT (creating a partner library), Us (grades, reputation), Companies with data(target customers)
Target market location: worldwide. Software technologies are unrestricted. But we will not offer non-English language support and if we include cryptographic technology, the software may be subject to export control.
Pricing: this is tough. Look at pricing that similar companies provide for their products with service agreements, e.g. Sqrrl.

NetBeans is soooooooo much better than Eclipse with m2eclipse plugin for Maven support.

From Graph computing article "As dataset sizes grow, it becomes increasingly important to choose algorithms that exploit the efficiency of **sequential** access as much as possible"
Argument to denormalize tables for sequential access, even at the expense of massive data size increase. Gain row locality, no additional lookups.
"the performance cost of storing and retrieving data on other nodes in a network is comparable to (and in the case of random access, potentially far less than) the cost of using disk."

23 September
My laptop: Quadro FX 880M Compute Capability 1.2
Ganesan's server: GeForce GTX 580 Compute Capability 2.0 (x3)
Sadly, my laptop's GPU is too old for Matlab Parallel Computing Toolbox. Matlab R2014a requires Compute Capability 1.3+
Ganesan's server is good to go.
demos here: C:\Program Files\MATLAB\R2014a\toolbox\distcomp\pctdemos
online:
http://www.mathworks.co.uk/help/distcomp/examples/illustrating-three-approaches-to-gpu-computing-the-mandelbrot-set.html?prodcode=DM&language=en

On Ganesan's server: can run NVIDIA GPU samples by copying /usr/local/cuda-6.0 to ~

SeniorD class 23 September 2014
Project proposal due 10/14
tiger.ece.stevens-tech.edu
Executive summary 200 words

How do we develop concepts? Identify the design decision tradeoffs and how we come to a final decision.

Example concept = choice of programming language

- Want high level language; don't want to write assembly
- Want language people are familiar with = fast startup time
- Want good GPU programming support with libraries

Which algorithms?

Simulate when analytic solutions not possible. For us we have the national laboratories to really test scalability.
Evaluate design: performance, how can we improve?

Risk: we don't get a performance edge. We think there is potential in GPUs and FPGAs but not sure. Also databases.
Recommendations: recommend seniorD be worth more credits. Next steps of project.

25 September
Data analytics industry emerging/growing - look at number of startups -
Not saturated; the best performance wins
We take the niche of GPU/FPGA computing

Growth trend positive in GPU/FPGA computing for companies that want the best, cutting edge performance, and can't get it using traditional computing

Fragmented market - no clear leading companies. Lots of technologies out there.

shared nothing architecture - no central memory store
NoSQL means does not strictly follow the relational model: column→value

Sacrifice consistency for performance

NewSQL tries for both consistency and performance

Plan for Ganean's server

- User umatlab, primary group gmatlab
- Change umatlab umask to 002 for user and group rwx and other rx
- Create new ssh key pair.
    - Add public to allowed_keys of umatlab.
    - Keep private in same directory, chmod 440, for extra security chown to root
    - Create /etc/profile.d/umatlab.sh with the line
      alias umatlab='ssh umatlab@localhost -i /home/umatlab/ssh/umatlab_ssh_priv'
    - Now other users in the same group as umatlab (gmatlab) can ssh in as umatlab
- Add all matlab-using users to gmatlab group

Demo schedule

1. Accumulo architecture
    a. 1.6 User guide

    b. Slides
2. Iterators
    a. Article with inline examples
3. Accumulo development with NetBeans
4. D4M
    a. d4mBB

Big result from meeting: idea to use a ScoreIterator in the Accumulo pipeline

TabletServer → ScoreIterator → MaxCombinerIterator → Client

The ScoreIterator does heavy computation involving Markov chains and such to compute a score for genetic sequences. The MaxCombiner only returns the row with the maximum score. GPUs can accelerate the ScoreIterator, called via JNI.

Xin, great to see the starting progress on a website. Good work!

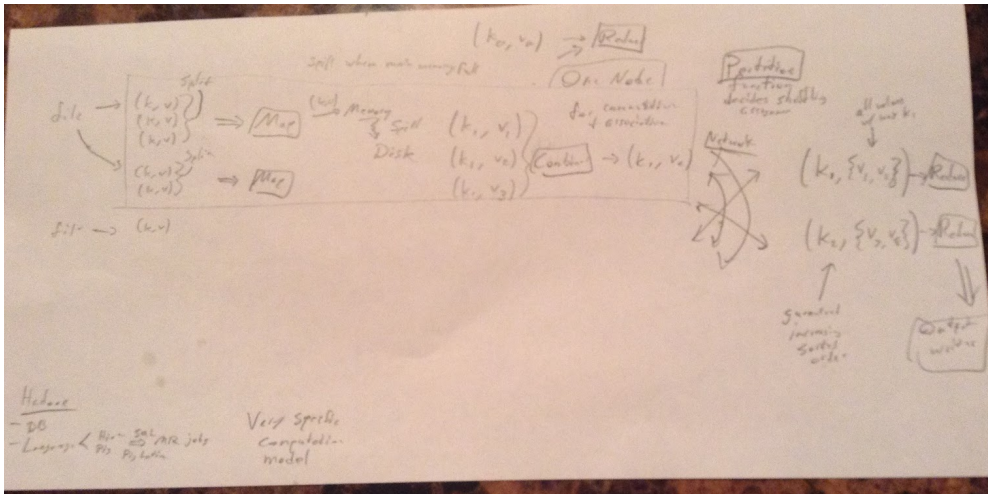
Yao, would you like to present the key findings of the GPU graph algorithm paper you read next week (and/or write something up about it)?

All, glad we're gaining an evermore clearer picture of the project. Please feel free to email me or Dr. Ganesan with any points of confusion. I can help with Accumulo in particular.

Matlab is now up and running on Dr. Ganesan's b412srv server. Please see the Ganesan server doc for info on how to connect. Di and Eric, you now have accounts on the server.

1 October
Random phrases
We target GPUs and FPGAs which they are unexploited; no one else has created a general graph analytics library for analytics on large data sets using them. This is due to their specialized nature. These analytics include sentiment analysis using Natural Language Processing techniques, community detection
We offer support to help them setup, use and maintain our library.

Read MapReduce survey paper:
A Survey of Large-Scale Analytical Query Processing in MapReduce
VLDB Journal 2014.
HDFS is best for sequential access, append-only, write-once/read-many



3 October

In case you were confused like I was about what graph partitioning is as we discussed at the end of Yao's talk, you can read about it here.

The idea is to divide a graph into k subgraphs such that the subgraphs are *good*. Usually we define *good* as there being many connections within each subgraph and few connections between subgraphs. This is community detection. You might imagine some other goodness measure.

By the way, one approach for graph partitioning is NMF Non-negative Matrix Factorization. This is a building block for topic modeling with k topics. The MIT group is currently looking into how we might implement NMF in the language of GraphBLAS. There's an *early* draft here.

Maven for eclipse
possible eclipse issue with JDK tools

The public Accumulo API is composed of :

* All public classes and interfaces in the org.apache.accumulo.core.client package, as as well as all of its subpackages excluding those named "impl".
* Key, Mutation, Value, Range, Condition, and ConditionalMutation in org.apache.accumulo.core.data.
* All public classes and interfaces in the org.apache.accumulo.minicluster package, as well as all of its subpackages excluding those named "impl".
* Anything with public or protected acccess within any Class or Interface that is in the public API. This includes, but is not limited to: methods, members classes, interfaces, and enums.

There is a MultiTableBatchWriter.
NamespaceOperations

listSplits

~~~~~
big data and big compute
~~~~~