Iterators great when amount of data going in > data going out of iterator.

What about the '<' case? Use case: parsing

> Suppose one cell in my row holds an XML document. I'd like to configure an iterator with an XPath expression to pull a field out of the document, so that I can leverage the distributed processing of the cluster instead of parsing the doc on the scanner-side.
> --Russ Weeks, 2014-08-29, rweeks@newbrightidea.com

Response

> there aren't strong guarantees about the lifetime of an iterator (so we wouldn't know when to close any resources held by an iterator instance, such as batch writer thread pools) and there's 0 resource management related to tablet server-to-tablet server communications.
> --William Slacum

Morale: don't rely on state in iterators. Don't rely on tablet server communication.

> I get that it's not supported or desirable for an Iterator to instantiate a scanner or writer. It's sort of analogous to opening a JDBC connection from inside a stored procedure - lots of reasons why that would be a bad idea.
> --Russ Weeks

Why? Cascading data reads/writes? Yes, seems bad, can't put finger on it.

> I'm more interested in the case where an iterator that processes input A, B, C, D might emit values A, A1=f(A), B, B1=f(B) etc. Under what conditions is it safe to use iterators this way? It seems there are at least two constraints:
>
> 1. A1 must sort lexicographically between A and B (otherwise the iterator could emit data out of order), and
> 2. A1 must be in the same row as A (otherwise A1 might properly be handled by a different tablet server).
>
> if you are returning a generated key that's not actually in the data, your iterator needs to handle the case where it is reseek'd with a range that has an exclusive start on a generated key. You'd have to potentially recompute results if you return multiple generated keys.
> --William Slacum
>
> Transforming data is tricky. You have covered some of the issues, such as making sure you generate sorted data within the tablets range. Also need to handle the reseek case. Accumulo reads batches of data. At any point it could take the last key the iterator returned and reseek non-inclusive. For example if you initially seek [A,R] and your iterator returns keys B,F,L,N,Q. You iterator should work correctly w/ the following seek ranges (B,R], (F,R], (L,R], (N,R], and (Q,R].
> --Keith Turner
>
> This expands on what Keith is saying. I have done something like this before.
>
> For this example, I am going to assume a few things. This is a mater of my opinion and experience and my not fall directly with what you want to do. I have experience with doing this where you insert into the same table, and insert into a different table. I am going to assume you want it in the same table, without needing to form a new connection.

Assumptions

You want to keep your original data
You are going to insert into the same table that you are reading from

Requirements

I am assuming some requirements, If there are others, please reply with them.

Data being read, must be output
This is a pretty straight forward requirement. Simple put that data you read, must also be a part of the output.

Order must be maintained (in most cases)
Ordering is very important in accumulo, in that Keys(rowId,cf,cq,access,time) must be in order. A small exception can be noted with scanners as currently its not enforced, however it is expected with ordered scanners. MINC and MAJC enforce the ordering policy.

Generated Data must come after the original record (in most cases)
Besides the exception with scanners mentioned before, ORDER MATTERS. So when you read a record, whatever you generate needs to come after it. This can be achieved by changing the column-family, column-qualifier, or accessors. **Do not modify the rowId as it cannot be guaranteed to be in the same tablet.** Also, changing the timestamp is used for accumulo's versioning so it is also best not to use that.

Implementation

Let us propose that the data we are generating is going into an *in order queue*. Additionally, the data being read is also in order. This allows us to create a **merge sort**, which means the resulting stream or dataset will also be in order.

Treating the data like this, resolves all of our requirements, granted that you create data that always comes after the generating data.

The complexity comes from all of the customization needed on the SortedIterator implementation.

It would also be recommended to run a versioning iterator BEFORE a generating iterator to limit the number records generated.

Pitfall
Memory: (Worst Case) memory can be an issue if the records are being output at the end of the tablet or scanner.
--Andrew Wells

Lookup exactly how iterators see data
What does **reseek** mean?

Check out org.apache.accumulo.core.iterators.user.TransformingIterator
https://github.com/fluo-io/fluo

> On 12/28/2013 12:53 AM, Dylan Hutchison wrote:
>> /1. Should the transpose table be built as part of ingest code or as an accumulo combiner?/
>> I recommend ingest code for much greater simplicity, though it may be possible to build a combiner to automatically ingest to a second table.
>> When inserting (row,col,val) triples, do another insert to the transpose with (col,row,val).
>> Use summing combiners to create the degree tables.
>
> Using a combiner is likely to be much more hassle than it's worth. *When your Combiner gets invoked server-side, you have no notion of lifecycle management and the only way to write to another table is to instantiate a Connector and BatchWriter.*
>
> As such, it's very difficult, and possibly impossible with the current API, to write to a separate table inside of a Combiner without leaking resources inside of the TabletServer.
>
> Definitely implement it in your ingest code :)
> --Josh Elser, 12/28/2013

ACCUMULO-1000 support compare and set

Use LargeRowIterator to suppress large rows? -- those supernodes with a gazillion connections

8 September
Accumulo User guide section 7.6
The Intersecting Iterator scans all tablets for RCVs that match ALL (set intersection) of a set of criteria, such as the resultant values all having the column families “the”, ”white”, “house”.

Row	CF	CQ	Value
BinID	Term	DocID	Weight
4344	the	unionAddr	1
4344	white	unionAddr	1
4344	white	unionAddr	1

(I’m pretty sure that all of one DocID rows must appear on the same shard (BinID).)

Thought: two tables in a pair:

1. In-table. RowID = NodeID, CQ=NodeID, Value=Weight. CQ-->RowID. Columns of adjacency matrix.
2. Out-Table. RowID = NodeID, CQ=NodeID, Value=Weight. RowID-->CQ. Rows of adjacency matrix.

9 September

Don’t return a row → deleted (for majc / minc)
Client-side iterator - not really used (dev stuff)
Tablet server iterators
Delete key -- everything returned afterward is deleted at compaction time

SortedKeyValue iterator
argument to init runs BEFORE you. Pull the next value from that.

Ilya
Iterator 1 reads the RFiles
Iterator 2
…
Iterator F Final iterator that sends data to the client.

1. F get called init which calls (F-1).init which calls (F-2).init … (2).init -> (1).init
2. SEEKs
3. F calls next → (F-1).next → … → (2).next → (1).next
4. SEEKs

Scan A-D, X-Z

TS1 A-C
TS2 D-F
..
TS3 W-Y
TS4 Z-Z

Iterator 1 looks at the RFiles

No backward
No save state

R1,CQ1
R1,CQ2

It2 next() → It1 next() returns A
It2 next() → returns modified form of previous - returns A’
It2 next() → It1.next()

Timestamp most recent first
Out of order returns ok in scanner but not at minc or majc

Directly modify the memtable? Uses server API, not guaranteed stable, could change on next version.

Locality groups - lookup by RowID AND ColumnFamily -- all stored on one RFile.

When at a specific tablet, scans are sequential search, not binary, because seeks are expensive. See Indexed Sequential Access Method (ISAM) file.

Todo look at Feature Vectors for Combining Iterators.

September 19

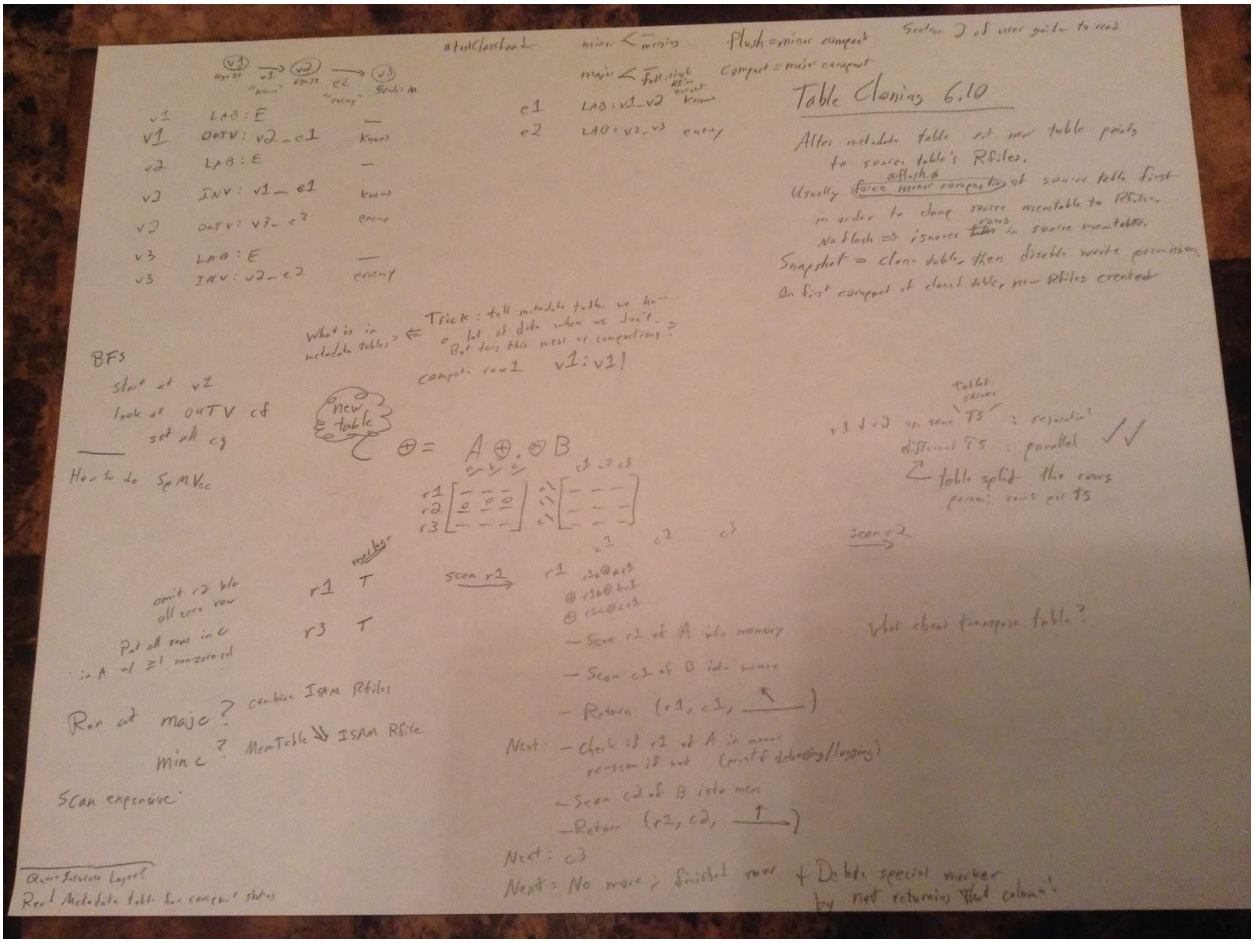
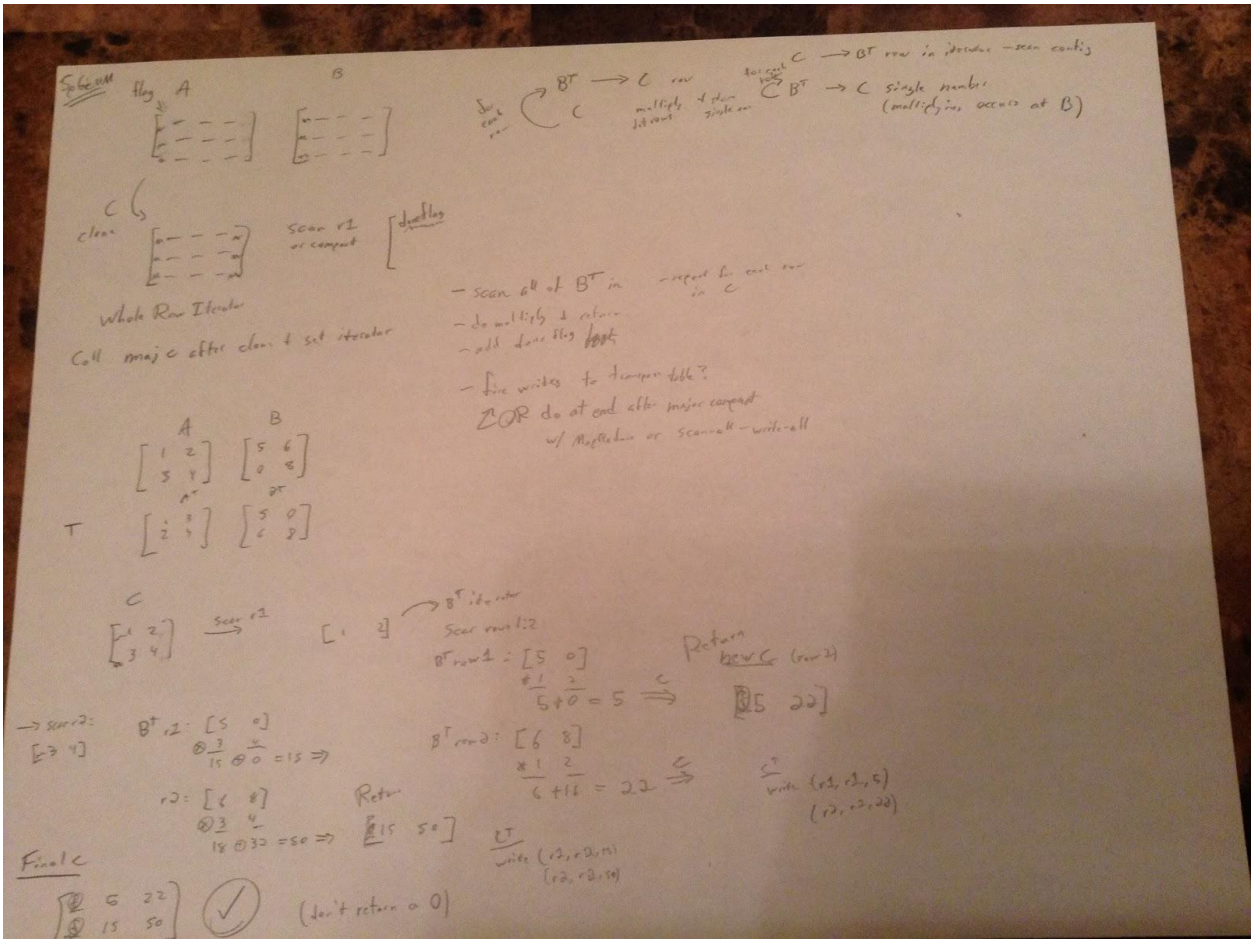


Table Clonings 6.10
Flush = minor compact
Compact = major compact
Scan 2 of user guide to read
v1 LAB : E
v1 OUTV: v2 - c1
v2 LAB : E
v2 INV : v1 - e1
v2 OUTV: v3 - e2
v3 LAB : E
v3 INV : v2 - e2
e1 LAB : v1 - v2
e2 LAB : v2 - v3
Snapshot = clone table, then disable write permissions
BFS
start at v1
look at OUTV of
set all c_j
How to do SpMVec
Trick : tell metadata table we have a lot of data when we don't. But does this mess up compactions?
compact: row1 v1 : v1!
new table
omit r2 b/c all zero row
Put all rows in C in A w/ ≥1 non-zero col
r1 T
r3 T
Scan r1
Scan r2
Run out major? combine ISAM RFiles
minor? MemTable ⇓ ISAM RFile
Scan expensive
What about transpose table?
Next: c3
Next = No more ; finished row + Delete special marker by not returning that column



19 September
Read this http://www.mammothdatallc.com/blog/accumulo-in-depth-look-at-filters-combiners-iterators-against-complex-values/
Nice email thread from Michael Moss <michael.moss@gmail.com> and William Slacum <wilhelm.von.cloud@accumulo.net> and Adam Fuchs <afuchs@apache.org> and Josh Elser <josh.elser@gmail.com>

Iterator Access
• Scan Time – Iterator has access to all keys that the user has authorizations to see
• Minor Compaction – Iterator has access to all of the keys that are in the MemTable – Iterator does not have access to keys stored in RFiles
• Major Compaction – Iterator only has access to keys that are in RFiles being compacted

How does MyPojoCountHistogramAll distinguish between keys of a table? The key returned is the last key in the table?

At this line in MyPojoCountHistogram, shouldn't we reset the countHistogram object in order to compute separate Histograms for each row? This looks like it will cummulatively compute the histograms. Want to study this in Accumulo.

```
ByteSequence expTsColFam = new ArrayByteSequence("expTs");

@Override
  public boolean acceptRow(SortedKeyValueIterator<Key, Value> rowIterator)  throws
IOException {
      while(rowIterator.hasTop()){
          //this is faster becaue it compares against byte arrays in key directly w/o converting to
string
          int cmp =
rowIterator.getTopKey().getColumnFamilyData().compareTo(expTsColFam);
          if (cmp == 0) {
              ....
          }else if(cmp > 0){
              //went past column family
              //can only do this w/ col fam since it next field in sort order after row
              break;
          }
          rowIterator.next();
      }

      return true;
```

from user list

Just in case you didn't know there is a 'classpath' command in the Accumulo shell which should list your custom jar. It's handy to verify that it was loaded. I think there might also be a log entry if you have access to them. I've also found it useful to use 'jar tf <filename> on the Accumulo nodes to verify the jar file contents. Sometimes I've deployed the wrong version of a jar file.

Actually, the classpath command won't verify if it was loaded, it will just verify that it should be loaded. There is a race condition in the file monitor that can cause jars in place to fail to be reloaded.

https://mail.google.com/mail/u/0/#search/label%3Amailers-accumulo++wholerowiterator/1422a91a14c85436

RowFilter vs. WholeRowIterator
Use RowFilter if the entire Row will not fit in memory. RowFilter has a trick when attempting to seek to the next row. Start by iterating the source (calling source.next()) up to 10 times. If we're still not at the next row, then seek the source to the next row.

```
    if (acceptRow(decisionIterator)) {
      currentRow = row;
      break;
```

```
} else {
  currentRow = null;
  int count = 0;
  while (source.hasTop() && count < 10 && source.getTopKey().getRow().equals(row)) {
    count++;
    source.next();
  }

  if (source.hasTop() && source.getTopKey().getRow().equals(row)) {
    Range nextRow = new Range(row, false, null, false);
    nextRow = range.clip(nextRow, true);
    if (nextRow == null)
      hasTop = false;
    else
      source.seek(nextRow, columnFamilies, inclusive);
  }
}
```

28 September
blog on installing Accumulo
Tracer based on Google Dapper
Rfile = relative key file. Indexed sequential access map file.
Accumulo internals chapter here at Safari.
Experiment todo: test the Rfile layout with locality groups

Idea: use a locality group on the column family for table + transpose.

Extension of idea: use many column families to have many versions of a graph
0001R
0001T
0002R
0002T

Framework of versions on entries in table. Suppose element-wise operations op1, op2, op3.
Suppose 5 rows of data

org.apache.accumulo.core.file.rfile.IndexIterator - at RFile
MultiIndexIterator implements FileSKVIterator
RFile < BlockFileReader <

MultiIterator at priority 8 holds our iterators, referenced by table property
Priority 1 majc iterator removes iterators

IntersectingIterator reamins within the same row.

row: shardID, colfam: term, colqual: docID
→ row: shardID, colfam: (empty), colqual: docID
returning only those docIDs that appear in all given terms
parallelize over all shardIDs

jps - Java Process Status Tool

class MatrixConnector extends Connector
void writeMatrix(String tn, Object[][] mat) -- Ingest function
Object[][] scanMatrix(String tn)
Object[][] scanMatrix(String tn, bool[] nodeSubset)
-- later

Write from a Constraint? -- BatchWriter in the check method of Constraint. from mail

> Sure, if you look at the interface on SortedKeyValueIterator, the only"lifecycle" type methods are init and deepCopy. In other words, you have some control over when to create a BatchWriter, but you don't know when Accumulo is going to tear down that iterator and stop using it.
> You could always create/use/close a batchwriter in an iterator without issue; however, it'll be difficult to keep a single BatchWriter alive for the desired lifecycle.
> In practice, Accumulo's lifecycle for a SKVI is either timeout related or related to how often the buffer of results between server and client fill. Normally, the case is when the buffer of results between server and client fills, Accumulo will tear down scan, and thus, your iterator.
> -- Josh Elser
> so we don't have control over when an iterator is destroyed...

Thanks again Josh.
The way I have been approaching it is to create/use/close the BatchWriter inside of the seek method when I need it. Do you see any issues with this approach?

> It's not terrible, but you will be incurring some extra overhead in thisapproach. The batchwriter is most efficient when you can keep a single instance open and just throw many mutations at it. Just make sure to close the batchwriter in a finally block, and you shouldn't have any problems.

Call me naive but why don't you know when Accumulo is going to tear down your iterator and stop using it? When I attach an iterator to a scanner, isn't it only destroyed after I complete my scan?

> You don't know because the SKVI API currently doesn't have any means to tell you. Yes, the tabletserver knows when it's about to, but you don't have means to be told this. This gets trickier with some of the work that Accumulo is doing under the hoods that I hinted at previously.
>
> Accumulo maintains a buffer between your (Batch)Scanner and the tserver(s) it communicates to. For a number of reasons, when that buffer fills up, Accumulo notes the last Key that scan returned, tears down your session, and (assuming the client is still there requesting more data), will then re-queue your scan to fetch more data starting back at where you left off.
>
> For example, if you have a table where each row is a letter in the alphabet, and you want to scan over all rows, you would just pass some range like (-inf, +inf). Suppose that after you return the letter 'f', that buffer fills up, and your scan gets torn down.
>
> Accumulo will restart your scan again with a different range than what you previously passed in: (f, +inf). This is an important note if you start doing "advanced topics" inside iterators that manipulate the Keys being returned, however it is relatively easy to work with.

What I have observed is something similar to the following....

```
init is called on creation
seek is called where you need to have the first K,V pair at the end of seek
hasTop, getTopKey, and getTopValue are called
next is called as long as hasTop is true
Once hasTop is false, the scan concludes
```

Idea: try and maintain a BatchScanner, close it in the Iterator class object finalizer.
If the BatchScanner is closed in between Iterator calls, then recreate it.
Assumes that the body of a method completes without interruption-- the method should not stop in the middle of the method..

BlackJack76 used DevNull to delete the data in a table. Better than BatchDeleter. Also retained table splits.

MiniAccumuloCluster starts and controls Zookeeper. Try it for testing.

Table Namespaces implemented but undocumented in Accumulo 1.6
Try to use accumulo-maven-plugin for integrating in Accumulo… in release notes

30 September

Read the Google Percolator paper. Built on top of Google BigTable - NoSQL DB with (Row,column,timestamp,value). Goal is to *maintain* an index. (I think this is the same as maintaining a *materialized view*.)

Note: use case is relaxed latency requirements. Some delay is okay (but not the days that MapReduce takes). The index will update eventually. Unlike real-time use cases, as in interactive querying, where latency must be minimized.

Google introduced Percolator because they didn't need to reprocess the entire data set after a single document is added. They wanted incremental computation.

Key architecture innovations

1. **Multirow transactions**. Use special columns to coordinate locking. Allows a transaction to read/write multiple rows atomically; either commit or abort. Fault tolerant due to logging in the special columns.

   Google needs multirow trans. for indexing web pages because they don't want to add a page to the DB without also adding it to the hashtable index. In the case of D4M, this guarantees that we update the degree table, regular table, transpose table, blob table in a single atomic transaction-- all or nothing. Google identifies which locks are critical and puts them in the table, and which are advisory, using a separate lower-priority lightweight lock service to improve performance.

   The guarantees of multirow transactions make client application programming much easier. Also, they took the thread-per-transaction approach with millions of threads, making each transaction blocking, as opposed to the non-blocking event-driven approach. They did this because they saw that a bottleneck was the number of RPC calls between clients and tablet servers.
2. **Observers**. Let Percolator clients scan over a special notify column. Rows mark the notify column when they update the value in a row. The Percolator clients scan over this column (in a different locality group for performance) randomly and constantly. When they see a row has changed since some observer last processed it, the client fires an event (RPC) to a registered observer for that column(one and only one gets invoked for a single column update). The registered observer can then call subsequent actions like more reads and writes.

   Google needs observers to handle multiple stages of incremental processing (without doing a batch process over all the data in a table).

- MapReduce -- for one-shot batch processing over ALL the data. No additional data structures. MapReduce will schedule tasks redundantly to help prevent "straggler shards" from slowing the whole pipeline.
- NoSQL DB -- indexes the data. Provides ability to scan targeted subsets of data. Single-row transactions. Low latency, super distributed.
- Percolator -- uses a schema and extra scanners at the DB to introduce multirow transactions and observers. Good for transactions that break down into small increments with as much parallelism as possible (aside from the ordering of the tasks). Some efficiency loss due to abort-restarts.

- SQL DB -- all the options you want-- ACID, serializable consistency, Joins, … not sure about latency but generally lower performance. Sign of a SQL DB: can it join first-class?

D4M case: simultaneous updates to degree table ok; there is a commutative combiner on the degree table. Writes to the normal and transpose and blob table don't conflict. The ingest is pretty much read/append-only. No need to guard against client failure like Percolator does. Also, we're online and interactive (I think). Need low latency.

Graph library case: we want to incrementally maintain the communities and such at the database. Register some listeners.

Can we set an iterator at write time to a DB? Maybe equivalent to minc time. Check consistency

Interesting:

- Snapshot isolation semantics. Equivalent to disabling the versioning iterator (or making it retain many versions and looking at the timestamp of all the rows in a transaction, only taking the state of the rows at the time of the transaction start.
- Roll forward concept. Thanks to the prewrite step, a client will finish the commit of another client that crashed after committing some of its data.
- Chubby lockservice on processes. Processes register with Chubby and automatically deregister on exit. Allows other processes to see if one has died or is still alive. Heartbeats ensure a process makes progress.
- Observer processes called as external processes from a binary that daemon-runs at every tablet server.
- While developing Percolator, Google modified the BigTable API to add conditional mutations, to reduce the number of RPCs.
- Prefetching built into Percolator somewhere. Caches rows that Percolator predicts will be accessed often by near-future transactions.
- Interesting questions
    - Infinite scalability in exchange for lower efficiency via adding more COTS machines?
    - Lower development time in exchange for lower efficiency via layered systems and APIs? ******
      "conventional wisdom on implementing databases is to...use hardware as directly as possible." Percolator puts an OS, several layers of software and network links between DB and the hardware. Cost in efficiency, but the layers allow infinite scalability, fault tolerance, and lower development time. Go for the generalist system, not the high-performance specialist!
- CAP tradeoff of Percolator: in order to increase C (consistency), they sacrificed some A (availability). Not as easy to replicate tables across data centers due to all these processes and state.
- Percolator design pushes back at this vision paper for the cloud, which says that transactional applications are hard to get in the cloud.

- Shared-nothing means no shared memory, no shared disks.

Proposal to use a Percolator-like design

Updates are made to the central adjacency matrix A frequently. Disable versioning. Iterator on A to keep the last value processed and the most recent value. Extra column with a bit vector, one for each column in the sdmvsdsvksdf details

If Frobenius norm gets too large, then rerun the NMF computation.

1 October

Looking at Fluo. How does it (and Percolator) handle high cardinality counts? Something about random column updates.

Looks like I can contribute to the fluo phrasecount example - look in the DocumentLoader and the PhraseCounter.

4 October

(-Inf,c], (c,g], (g,w], (w,Inf) as long as I'm remembering the inclusivity correctly.

One tiny correction: by exclusive -Inf, we mean inclusive empty byte array.

Given table name,

- get a list of all tablet servers by connecting to the Master and referencing the MasterMonitorInfo
- get internal table ID via Tables.getNameToIdMap
- connect to each tablet server TabletStats of tablets that are on the tablet server under the given internal table ID
- Scan Metadata table starting at the {tableName converted to internal table ID}
- and ending at {internal table ID}'<' (last entry for this table in the metadata table)
  - Example row: 1< (if the internal table ID is 1 and this is the last split in the row)
- look at the column for the previous row: ~tab:~pr
  - Example row-col-val: 1< ~tab:~pr [] \x00
  - (this table has no table splits-- no end row and no previous row start)
- Create an extent for the value using KeyExtent
  - (shortcut for parsing the metadata table and getting the previous and current end row)
- Among the list of TabletStats, find the one whose previous end row and next end row match the result from the Metadata table.
- Take that **tabletStat.numEntries** to get the number of entries in this table split range.

Later this information is combined into a method that returns an array of triples

(tablet_split_range, tablet_num_entries, tablet_server_list_for_this_tablet)

org.apache.accumulo.core.Constants.METADATA_TABLE_NAME = "!METADATA"
org.apache.accumulo.core.Constants.METADATA_PREV_ROW_COLUMN = new ColumnFQ(new Text("~tab"), new Text("~pr"));

function [splitString, varargout] = getSplits(T)
%GETSPLITS gets the current splits for a table

```
%   OUTPUT splitString: the splits of T in the format f,p,r2,w,
%   OPTIONAL 2nd OUTPUT: a comma-delimited string that holds N+1 numbers
%   where N is the number of splits and the (i)th number is the number of
%   entries in tablet holding the (i-1)st split and the (i)th split.
T_s = struct(T);
DB_s = struct(T_s.DB);
javaOp =
DBaddJavaOps('edu.mit.ll.d4m.db.cloud.D4mDbTableOperations',DB_s.instanceName,
DB_s.host, DB_s.user, DB_s.pass, 'Accumulo');
if nargout <= 1
   splitString = javaOp.getSplits(T_s.name);
elseif nargout = 2
   %user wants the count in each tablet
   retArray = javaOp.getSplits(T_s.name, true);
   splitString = cell2mat(cell(retArray(1)));
   varargout(1) = cell(retArray(2));
else % also return the tablet server names that each tablet is assigned to
        retArray = javaOp.getSplits(T_s.name, true);
   splitString = cell2mat(cell(retArray(1)));
   varargout(1) = cell(retArray(2));
        varargout(2) = cell(retArray(3));
end
```

Need to figure out the return methods of these guys.

[To read Field Cardinality and HyperLogLog]