Casebook1 (Masoud Moshref Javadi)

Max user: 30 (1000(*2K)posts, 100 photos) **course**: CS694 **Address**: http://204.57.0.195

- **Architecture**:
  - 4 Cassandra nodes and 1 DualCore front node.
  - Column families: Tweet, Photo, User, Timeline, PhotoAlbum, PhotoData, Friends
- **Load algorithm**:
  - Load config file
  - Iterate over all lines and create queries that writes a row in the output with only one command: For example, create the list of photoIDs of the user and just by one RPC command commit the photoalbum of the user.
  - Create multiple threads per each column family and run them
  - Wait for threads
- **Refinements**:
  - Each thread uses a Batch Mutator which caches queries/updates and send them in one request to the Cassandra node. Each time the query cache fills up a connection will be picked up from the pool (connection pool size is equal to the total number of threads). If the size of this cache is large it takes larger memory and more time to fill it so bandwidth usage will be low.
  - Each Batch Mutator will be replaced by a new one when changed 10 connections.
  - Set Initial Token of ith Cassandra node to $i * 2^{127}/4$ to have equal load on each server (the default setting balances the load with 50%, 25%, 12.5% and 12.5% shares based on the order of server startup.
  - Select the number of threads per column family based on the size of its data.
  - Not generate IDs randomly, UUID generator takes time, just generate it from line number
  - To show photos, they will be downloaded from Cassandra and cached on the webserver (this cache will be cleared on reset)
- **Discussion**:
  - Why DualCore front node: Because with threading, I could use multiple cores. Now up to 150% CPU usage. Besides, it has 1Gb memory that I need for batch commits. But for Cassandra nodes, the CPU usage is not enough to use DualCore
  - How long do the first sorting and query creation take? Less than 1 second. But download time is large and I did not know that load time is included in the time. I could load 45 users with local file
  - Is that better to commit data as you are sorting them? No 1) 1 second is not so much. 2) thread changing elongates execution. If we consider load time it may work
  - As each Cassandra node may forward the traffic to the another node based on the MD5(row ID) and Token of node, is that better to use another simple mapping (other than MD5) and send traffic from front node directly to that node? No, as the input/output bandwidth of each node is not full now and load balancing in the direct mapping is hard
  - Which data takes longer to save? Timelines as they contain lots of information 11*(1100)=12100 timestamps and ids

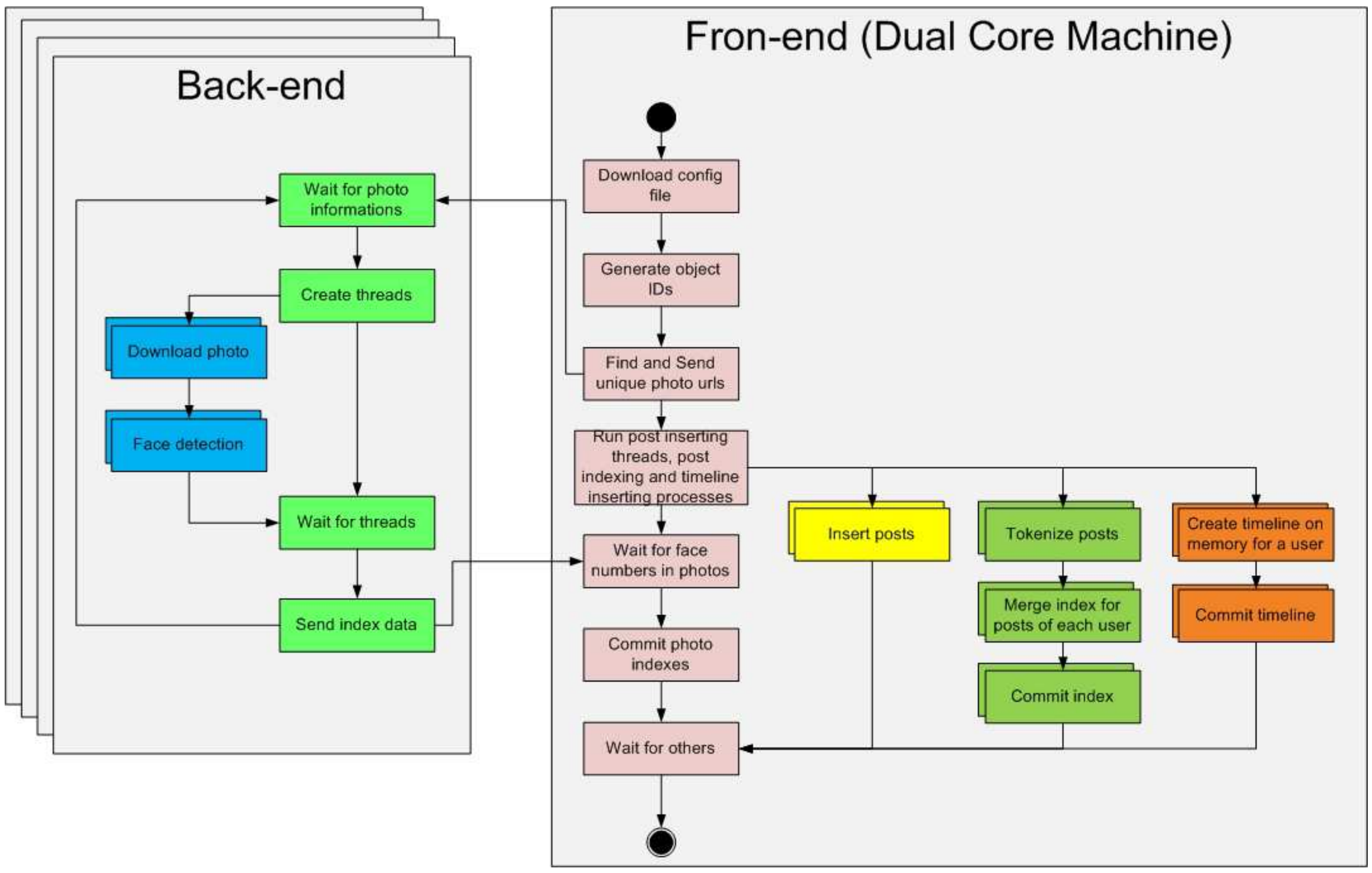
Masoud Moshref Javadi 5353968206 12/02/2011

Casebook2

Maximum number of users =140 website: http://204.57.0.195

- **Structure of Data**:
  - Data structure has been changed since part1 to embed object type (post, photo) and timestamp in ID.
  - The column families are as follows (rowid, column:data):
    - **User**: user ID, 'Pass':password, 'name':Name
    - **Friend**: user ID, (friend ID, timestamp of friendship)*
    - **Photo**: user ID, (photo ID, Photodata ID)*
    - **Post**: user ID, (post ID, post data)*
    - **PhotoData**: photodata ID, 'data': photo data
    - **Timeline**: user ID, (post/photo ID: poster ID)*
    - **PhotoIndex**: user ID,(Face number: list of photo IDs)*
    - **PostIndex**: user ID, (word: list of post IDs)*
- **Deployment**
  - Four Cassandra nodes with 1 core and 1 GB RAM and one 2 core machine with 2 GB
  - Photo downloading and indexing is also done on Cassandra nodes (see Figure 1)
- **Questions**:
  - Which Face detection package did you use? OpenCV
  - How did you tune it? It is tuned for speed not accuracy but I checked it with more accuracy and did not get better result in current data set. Someone may want to train it first then use
  - Why did you put photo indexing on backend nodes? Because face detection is a CPU bound application and back-end CPUs where unused. Also it helps download photos distributed.
  - Why not put other functions like post indexing or timeline commit there? They are not CPU bound, we need to send much data to back-end nodes and for 490 photos in current dataset we need almost 90 seconds for face detection so the new arrangement does not help so much.
  - What is the bottleneck then? I got full CPU usage on back end machines during face detection (Figure 2) and almost full CPU usage in the front end one. So for backend machines, bottleneck is CPU while for the front-end one while the CPU usage is high, it seems that network or Cassandra nodes cannot handle more (Figure 3).
  - How did you use multiple cores using python? I used muli-process programming instead of multi-threading
  - How does to not create post index for queries with scope 'snet' and 'glob' affect their performance? With profiling we can see that only 19% of time is consumed by query code while the remaining has been used by django to render the template. Having short delay to present queries with rarely used words approves this claim too that query time for index of posts does not have much effect on this delay.

- How much time does it take to download the file? About 10 seconds. Does multi-segment downloading help reduce this time? No



Back-end
Wait for photo informations
Create threads
Download photo
Face detection
Wait for threads
Send index data
Fron-end (Dual Core Machine)
Download config file
Generate object IDs
Find and Send unique photo urls
Run post inserting threads, post indexing and timeline inserting processes
Wait for face numbers in photos
Commit photo indexes
Wait for others
Insert posts
Tokenize posts
Merge index for posts of each user
Commit index
Create timeline on memory for a user
Commit timeline

Figure 1: Structure of Program and its deployment on machines

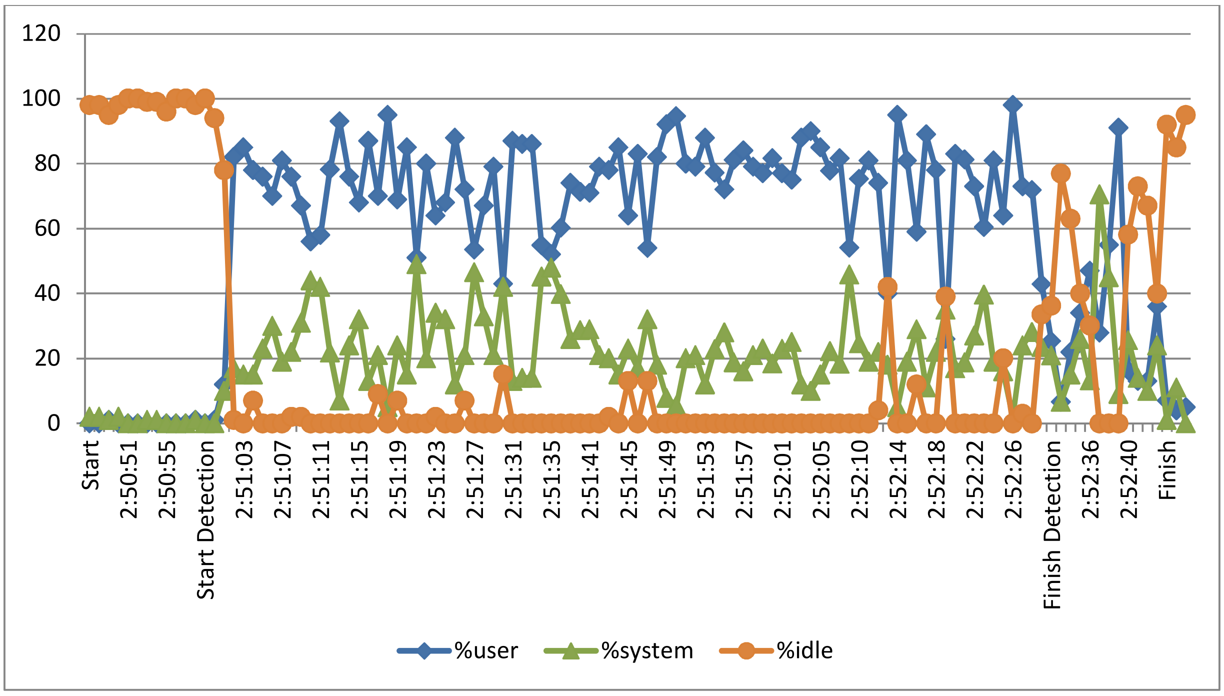


Figure 2: CPU usage in back-end nodes

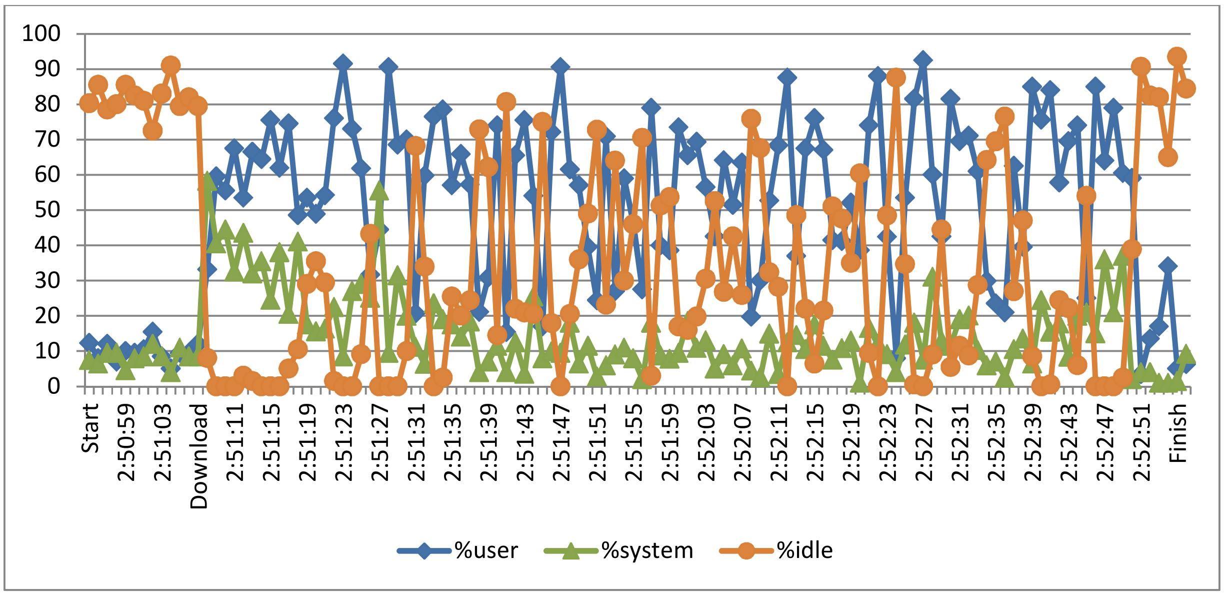


Figure 3: CPU usage in front-end nodes