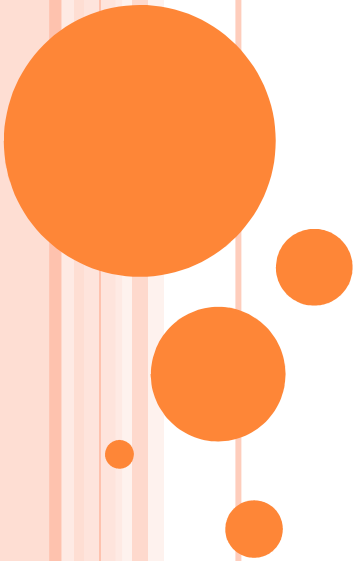
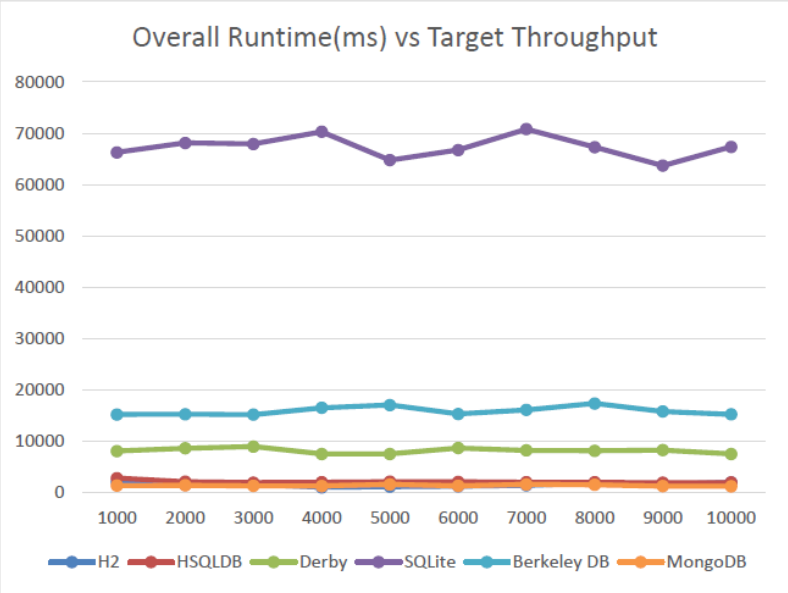


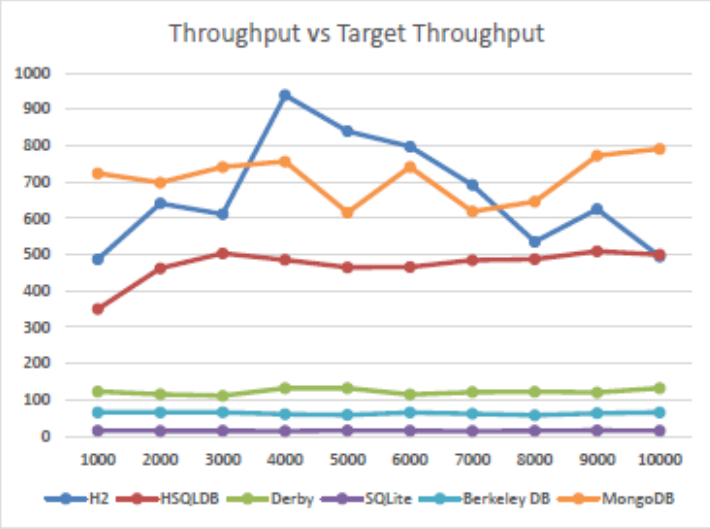
Embedded Databases MicroBenchmark

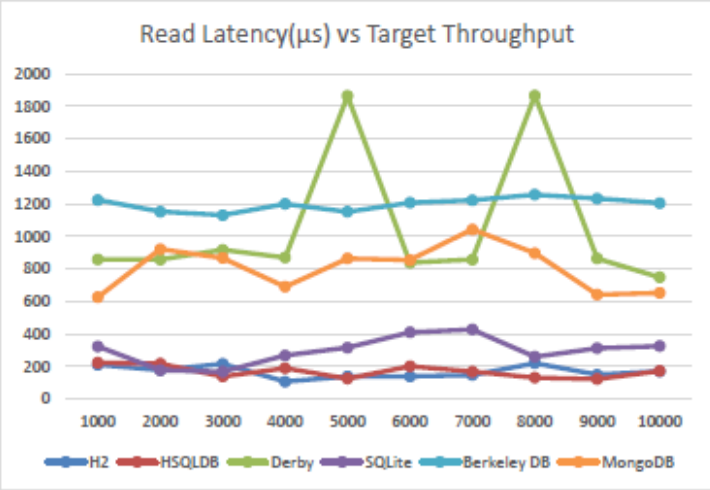
Team CodeBlooded

YCSB Results – WorkLoad A

Workload A (50% reads, 50% updates)

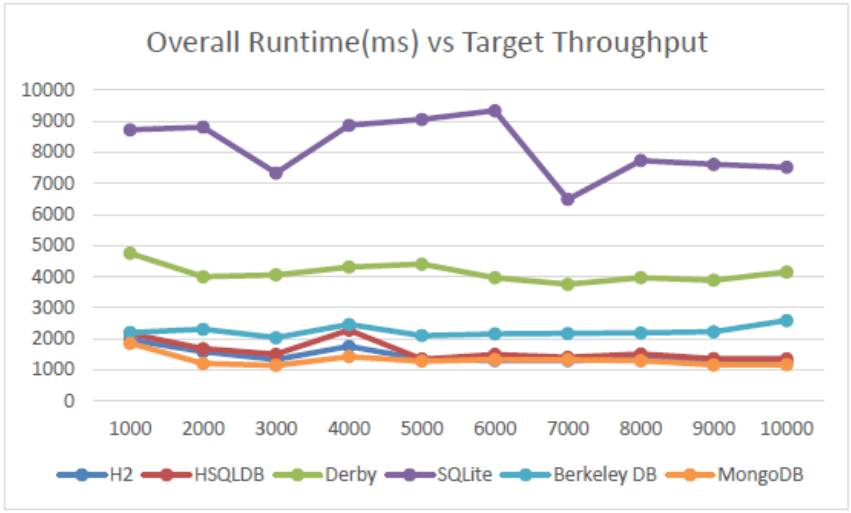


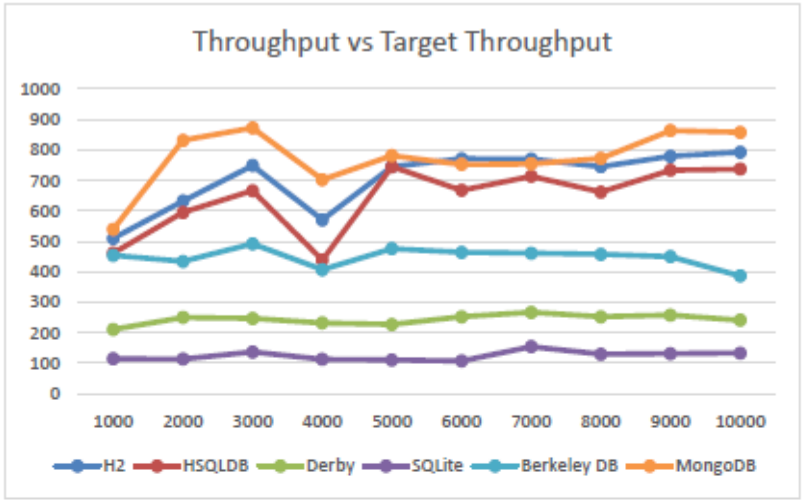


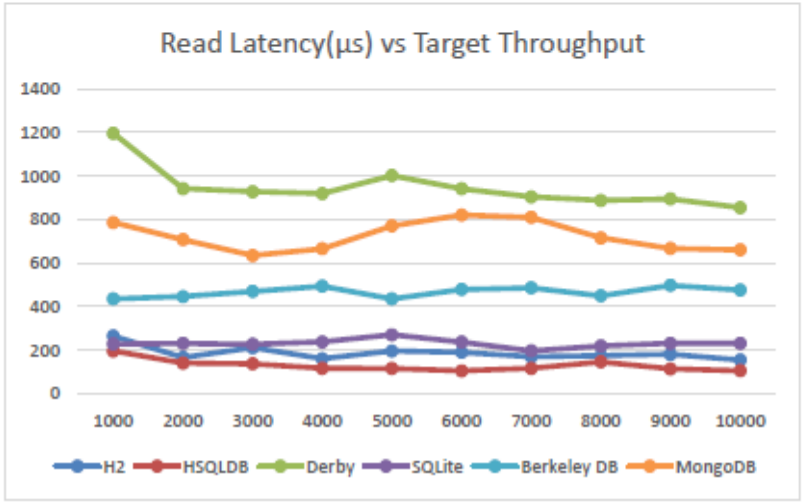


YCSB Results – WorkLoad B

Workload B (95% reads, 5% updates)

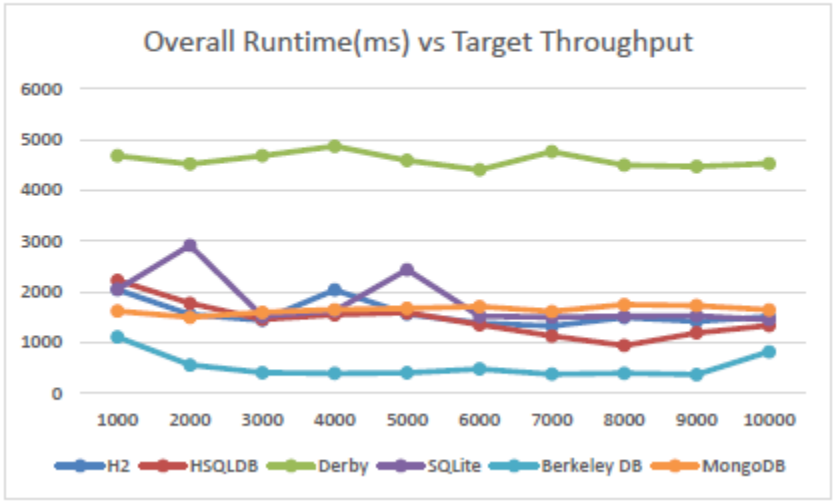


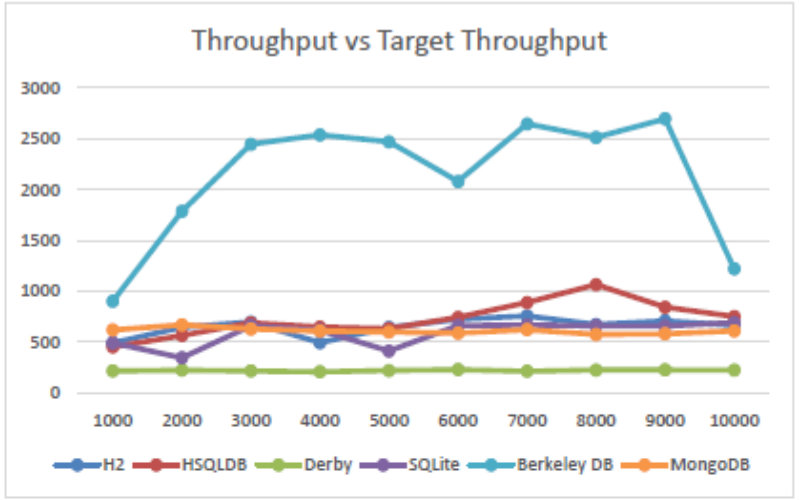


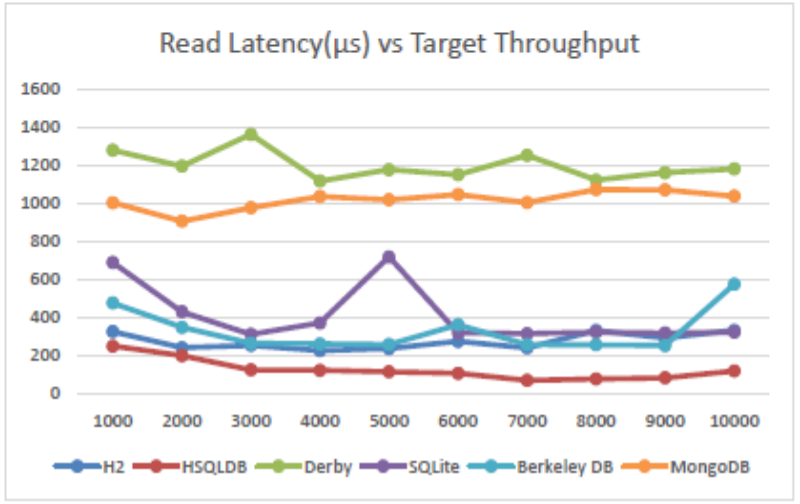


YCSB Results – WorkLoad C

Workload C (100% reads)







Observations

- SQLite showed better performance in the very first run but was seriously lagging by the third run
- Berkley DB performs best for read-only workloads and is somewhere in between and performs average on other workloads.
- H2 and HSQLDB are neck and neck in workloads A and B. In read-only workload, HSQLDB seems to be the better of the two.

Observations

- Apache Derby seems to perform poorly on read-only workload. It is not very convincing on the others either.
- MongoDB seems to perform well on all 3 workloads.

Why another benchmark?

- YCSB is general in nature – does not address embedded databases or their applications
- No true independent comparison of embedded databases
- Need for a benchmark that understands embedded databases and what they aim to do

The new Benchmark

- Having tested on available workloads, we now plan to develop our own micro-benchmark for embedded databases. The Final deliverable for the project comprises of 3 major sections:

  1. The Test suite
  2. The Connectors for the databases and
  3. The Workloads

The new Benchmark

Workload	Target Application Type	Target Application Example	Features
A	Read only persistence	Programmed devices, Caches	100% Reads
B	Key value stores	Mobile apps, Browser cookies/Bookmarks	50% Reads, 40% Inserts 10%Updates
C	Desktop Media Applications	iTunes, photos	70% Reads, 20% Inserts, 10% Updates
D	Internet of things	Sensors, Cameras, Id Scanners	80% Complex Inserts (datatypes such as blobs/clobs), 20% Complex Reads (joins, aggregate functions)
E	General Purpose Applications	Version/Source Control	50% Reads (with joins), 30% Inserts, 20% Updates

Table 1: Workload Classification.

Next Steps – Design Features

- In the initial phase of implementation we will aim to measure the Runtime, Throughput, Latency and Memory consumption at intervals of time.
- Why? - Performance & utilization of memory are key characteristics of embedded databases
- At the end of a test run the Test Suite will log the results of the current workload and generate csv files with recorded results

Lightweight Runtimes

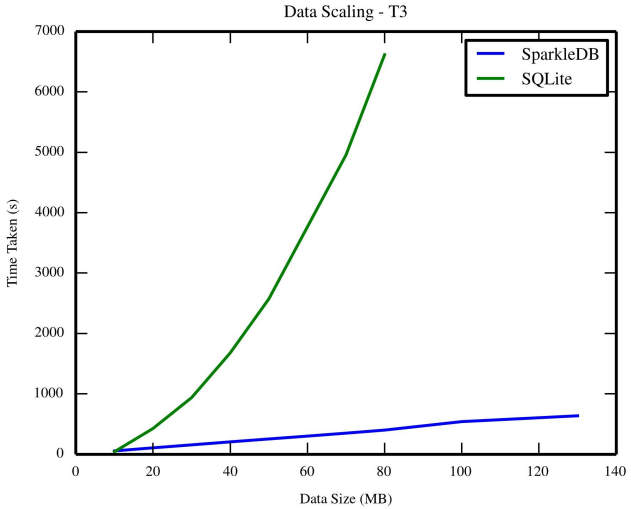
Team Sparkle

Dhinesh
Shiva
Keno
Guru

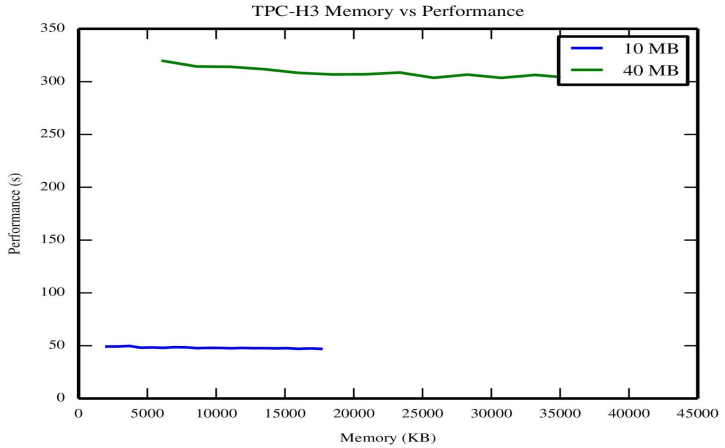
The goal again

- Understand characteristics of our intended workload (streaming data)
- Study capabilities of currently available IoT hardware (Intel Galileo)
  - Intel Quark 32-bit CPU
  - 400MHz processor speed
  - 256MB DDR3 memory
  - Debian Wheezy (7.9)
    (This is the setup for all experimental results presented later)
- Design a lightweight query processor from an existing implementation (562)

Evaluating what we currently have

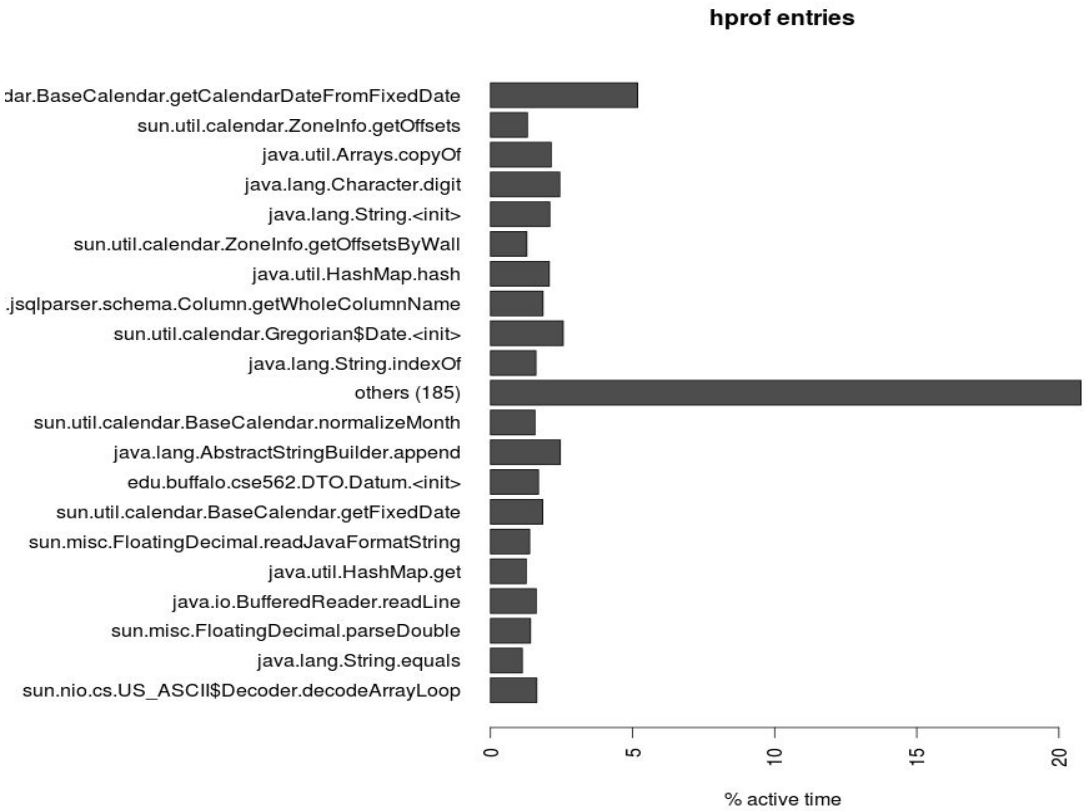


TPCH-3 with increasing datasets, in comparison with SQLite



TPCH-3 with increasing heap size allocation

Identifying Bottlenecks



hprof cpu samples for TPCH-3 on a 50MB dataset

Narrowing Down

- Thermostat
- Possible stream sources:
  - Tracking entry and exit from rooms
  - Logging room temperatures
- Generate and evaluate queries that join and aggregate over both streams

Leaning towards aggregations for which some level of accuracy is expected, but not necessarily an exact result

Other similar use-cases would also be considered

Necessary Modifications

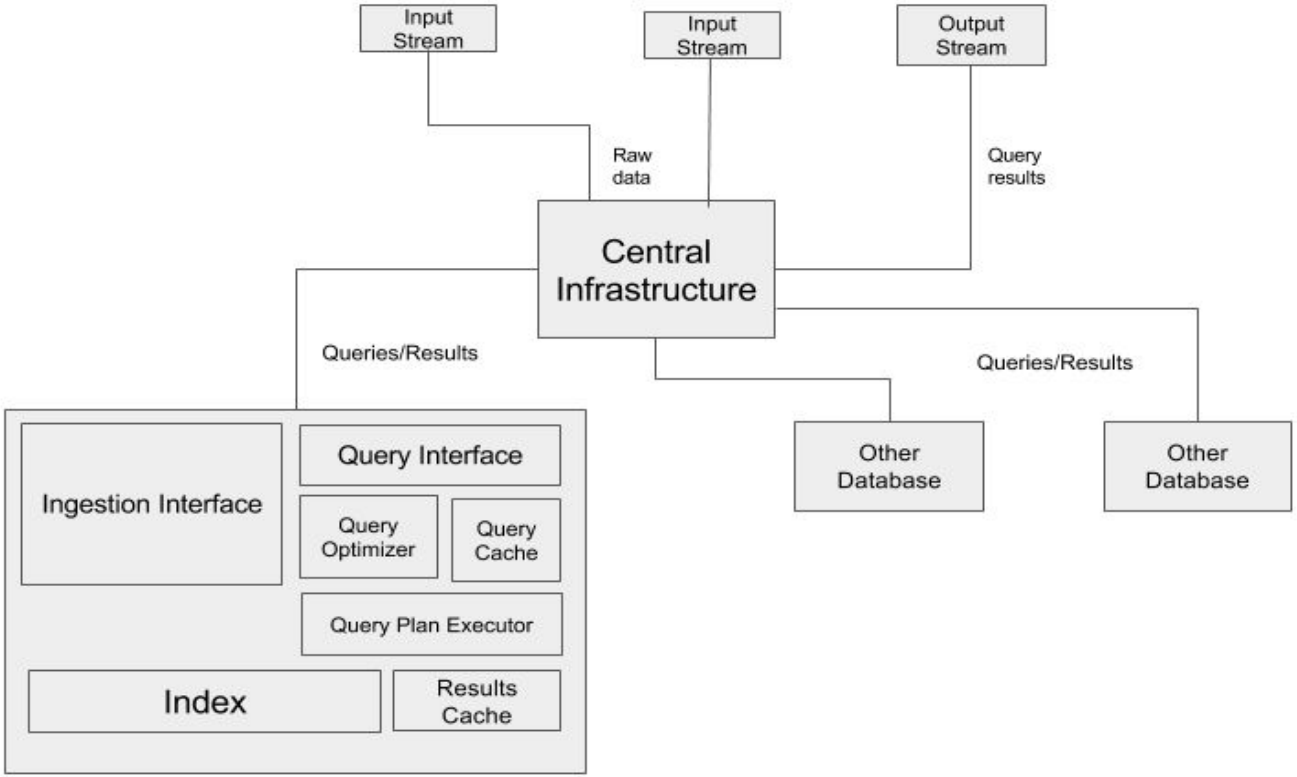
- Query optimizer that takes into account streaming data characteristics
  - Stream frequencies
  - Window sizes
- Query interface that allows only a subset of SQL queries relevant to us
- An ingestion interface, which also helps track stream frequencies

A Future Evaluation Strategy

Given two streams,

- Run query requiring a join over both streams
- Evaluate percentage of expected results we produce over given window
- Repeat with increasing stream frequencies
- Repeat with increasing window sizes

Moving Forward, and the Big Picture



PocketData

Naveen, Sankar, Saravanan, Sathish

What did we do last week?

Completed

- Extracting joins.
- Extracting where clause conditions.
- Aggregate functions.

In progress

- Extracting data from sub queries
- Normalizing data (like converting join on to where class conditions)
- Cluster Analysis

Our findings vs TPC Mobile paper

Mostly agrees with TPC-mobile paper.

Except few cases like MAX.

- 62769 vs 314,970
- We did not account for aggregate functions in subqueries
- MAX is commonly used in subqueries of form “Give me biology class 1st rank holder’s all subject marks”.

Cluster Analyzing

- Started analyzing apps from “real time notification” cluster.
- What does a real time notification app do?
  - Runs a service 24 x 7 unless battery saver is on to “poll” the server for updates and notifies user in realtime.
- We analyzed read write ratio of 8 apps: (Twitter, Facebook, Messenger, Google+, Hangouts, LinkedIn, Whatsapp, Pinterest)

Google+ vs LinkedIn

Similarities:

- Both social networking apps
- Both have features like posts, likes, comments, publically sharing posts, follow, connections, news feed etc.,
- No one uses any of these two.

Differences:

App	SELECT	INSERT	PRAGMA	DELETE	UPDATE
Google+	1384356 (83.67%)	24898 (1.5%)	150477(9.09%)	5226 (0.31%)	89391 (5.4%)
LinkedIn	431 (4.29%)	7221(71.97%)	610 (6.07%)	1478 (14.72%)	294 (2.93%)

Why so much difference in select and insert?

- Google+ being used as a content provider for other related apps.
  - Profile pic & profile URL for contact, gmail, hangout etc.,
- LinkedIn is independent.
- The more an app is used/opened the more SELECT queries count is. In Google+ the users are other apps.
- If the app is hardly used, insert queries percentage will be very high.

What is the inference?

Number of SELECT query Vs INSERT query ratio is highly correlated with usage.

Hangout Vs Whatsapp

Similarities:

1. Instant messaging apps
2. Stores notification against each contacts with recent messages, sorted by most recent message.

Differences:

App	SELECT	INSERT	PRAGMA	DELETE	UPDATE
Hangouts	631989(55.9%)	109006(9.64%)	175766(15.54%)	77892(6.88%)	135892(12.02%)
WhatsApp	75698(55%)	2734(1.98%)	22390(16.26%)	6603(4.79%)	30192(21.93%)

Hangout Vs Whatsapp

Updates:

- Updates unread notification against each contact.
- Updates the sorted by timestamp for each new message.
- Updates recent messages.
- Updates each contact that the user is available for new options like calls.

For a heavily used instant app,
Select > updates > inserts

Inference:
Heavily used instant message app with normalized tables will have huge percentage of updates.

Next step

Finding patterns of access within a cluster.

Parallely work on parser to extract more data.

LLVM Query Runtime

VALKyrie

Arindam
Kaushik

Ladan
Vinayak

Progress

LLVM Prototypes

- Accessing variables declared in C land
- Dereferencing pointers
- Passing function arguments

Started work on implementation -

- Schema
- Basic table materialization

Review of Our Challenges

- Dealing with immutability that SSA entails
  - LLVM does require all register values to be in SSA form

- Accessing variables and functions defined outside LLVM
  - Given an array defined in C, print its elements

Dealing with immutability

- Phi Node
  - Use for branching (If-Else) and simple loops

- Alloca
  - LLVM does not require (or permit) memory objects to be in SSA form
  - all memory accesses are explicit with load/store instructions

Accessing variables in C land

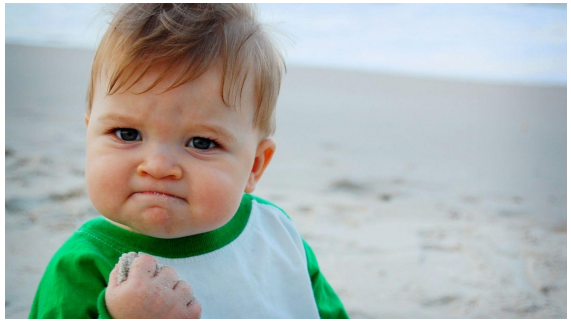
- Array.c

```
int array[5] = {0,1,4,9,16}
```

- Looper.cpp

```
Constant *conArray = module->getOrInsertGlobal("array", ArrayType::get(intType, 5));
Value* indices[2];
indices[0] = ConstantInt::get(intType, 0);
indices[1] = i;
ArrayRef<Value *> indicesRef(indices);
Value *v = builder.CreateLoad(builder.CreateGEP(conArray, indicesRef, "arr")
```

```
$> llc  dump.ll -o dump.s
$> gcc -std=c99 dump.s array.c -o finalprog
$> ./finalprog
0 1 4 9 16
```



Passing function args

```
// Calling LLVM function “print” from C land
print(array, SIZE);
```

```
// LLVM function
auto args = printFunction->arg_begin();
Value *array = args++;
Value *size = args++;

Value* indices[1];
indices[0] = i;
ArrayRef<Value *> indicesRef(indices);

Value *tmp = builder.CreateGEP(array, indicesRef, "array");
```

Next Steps

Implementing the Scan operator

Start looking into the Java/C++ interface

- Challenges
  - Mapping expressions to a good representation in c++
  - Accessing structs and unions defined outside LLVM