Embedded Database Benchmark

Team CodeBlooded

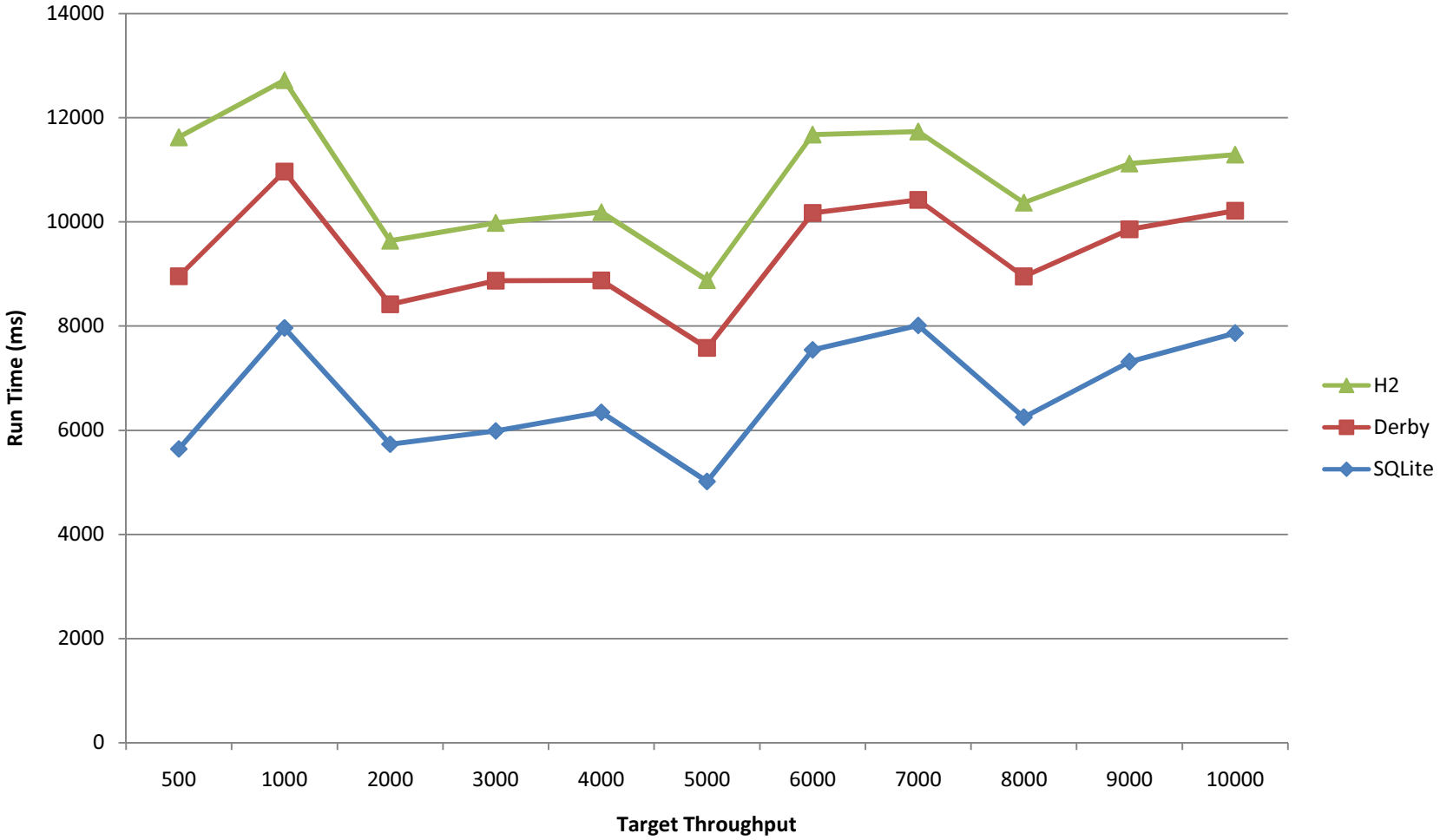
Weekly Report

- YCSB connected with H2, Derby and SQLite.
- HSQLDB benchmarked last week was in server mode.
- Still working on HSQLDB (embedded) and BerkeleyDB connection.
- Benchmarking with loads A (50% reads and 50% updates) and B (95% reads and 5% updates)
- Went through more papers/material on existing or on-going projects in benchmarking like OLTP-bench

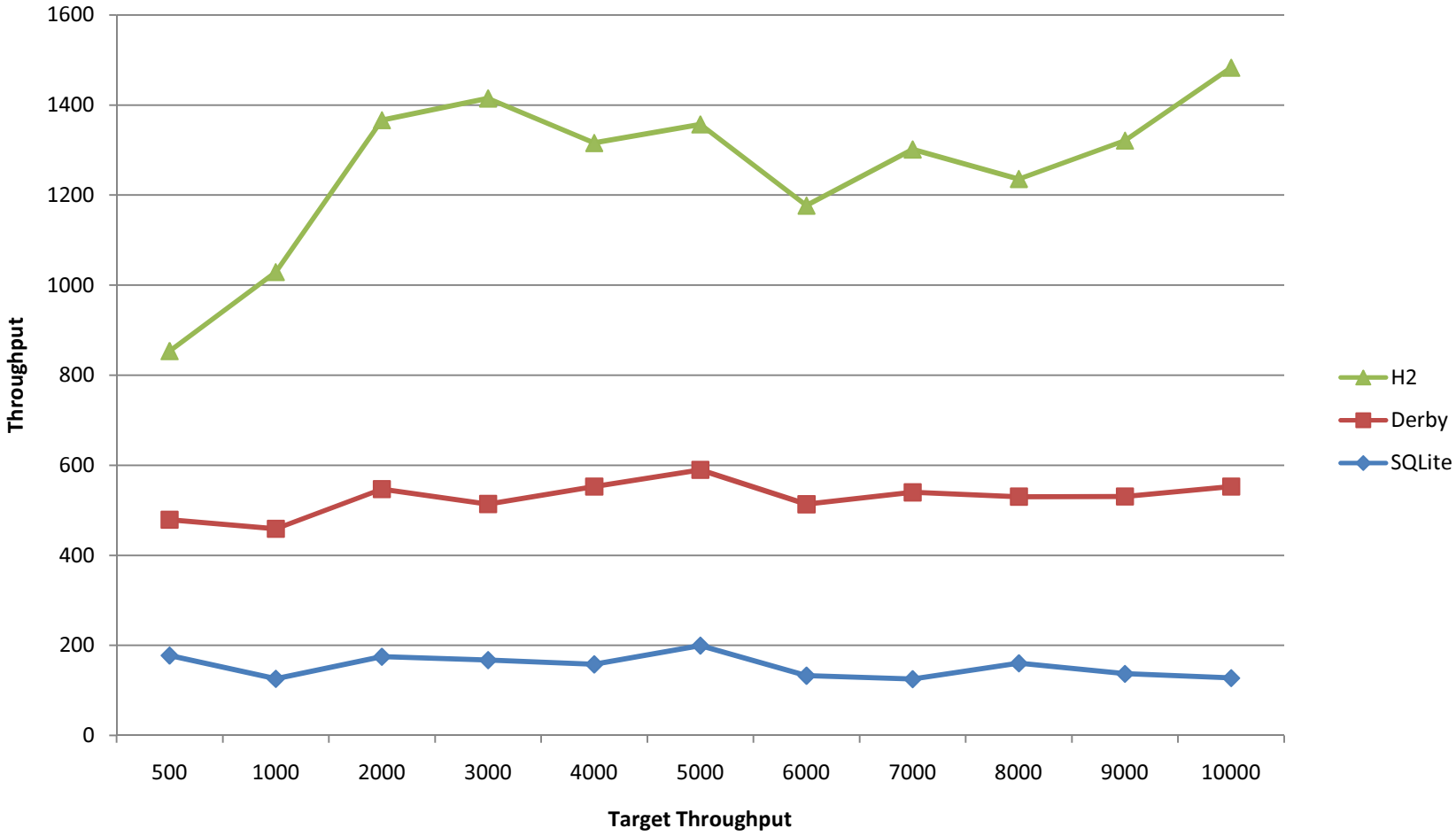
Initial Observations

- H2 gives better throughput at the cost of runtime.
- SQLite throughput was surprisingly low.
- SQLite has locking issues in multi-threaded runs. Much poorer than what we expected

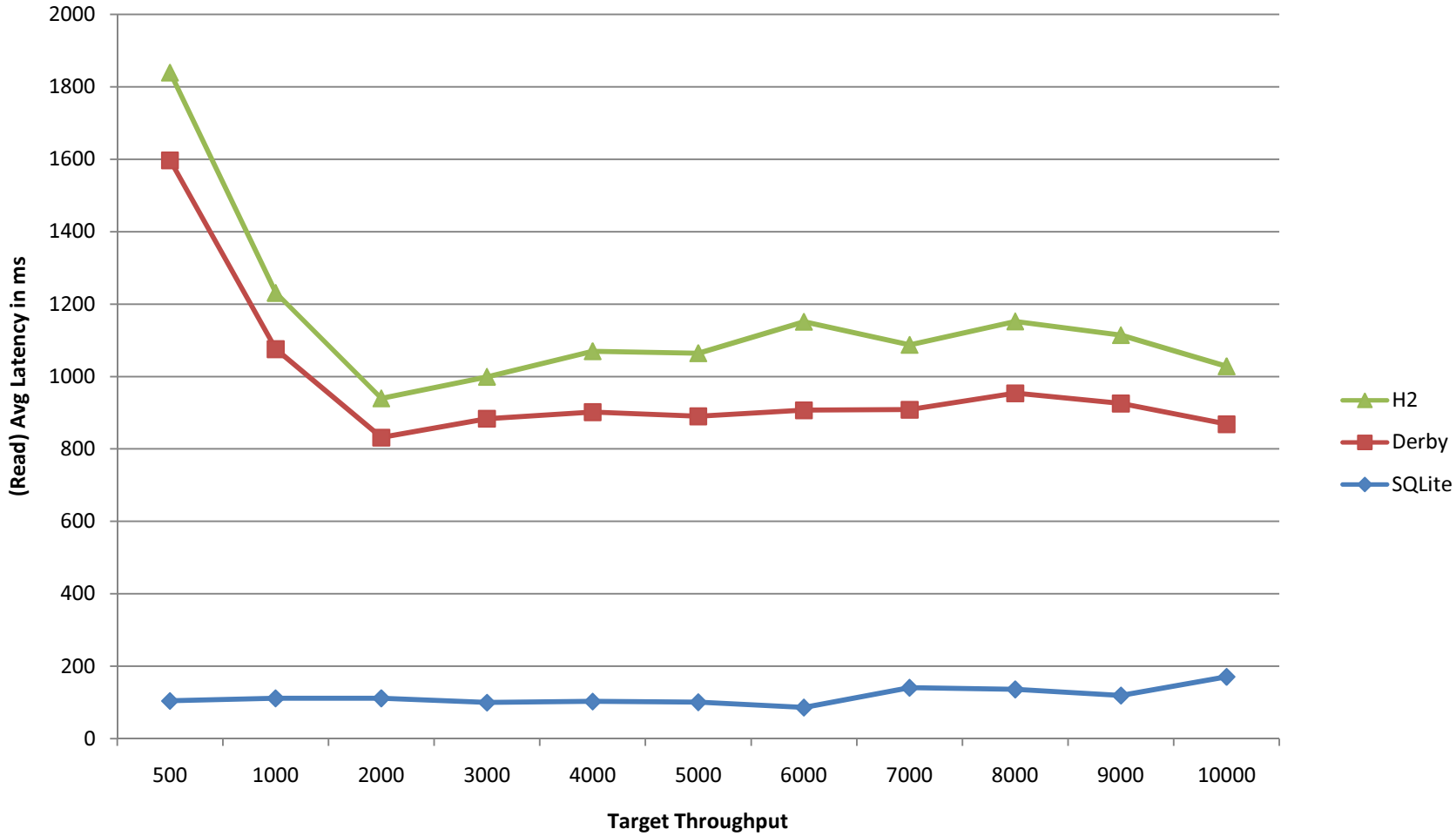
Run times


14000
12000
10000
8000
6000
4000
2000
0
Run Time (ms)
500
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Target Throughput
H2
Derby
SQLite

Throughput


1600
1400
1200
1000
800
600
400
200
0
Throughput
500
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Target Throughput
H2
Derby
SQLite

Read Latency



Notes

- YCSB has several issues with Windows
  - Our benchmark should be OS agnostic
  - Does OS play a role in the DB performance?
- NoSQL DBs can also be used in embedded form
  - Should we take them into consideration while designing benchmark
- Can we go beyond benchmarking?

Questions?

PocketData Benchmark

[Week #2]

Naveen, Sankar, Saravanan, Sathish

Progress

- Classification based on business domain.
  - 173 applications.
  - 26 clusters
- Finding features (In Progress)
  - Read & Write percentage,
  - Bursts,
  - Complexity of queries etc.

26 Clusters

System Services Utility Music Pre-Installed Networking

e-Commerce Messaging & Calls Game

Cloud Video Services Browser Image Forum New/RSS Reader

Cloud Storage Email Search eReader Dating

Navigation Media Player | Finance Health | Programming | Productivity | Personalization | Antivirus | Miscellaneous

Observations from 'Pocket Data: The Need for TPC-MOBILE' paper

Types and numbers of SQL statements executed during the one-month trace

Operation	SELECT	INSERT	UPSERT	UPDATE	DELETE	Total
Count	33,470,310	1,953,279	7,376,648	1,041,967	1,248,594	45,090,798
Runtime (ms)	1.13	2.31	0.93	6.59	3.78	
Features Used						
OUTER JOIN	391,052				236	391,288
DISTINCT	1,888,013			25	5,586	1,893,624
LIMIT	1,165,096				422	1,165,518
ORDER BY	3,168,915				194	3,169,109
Aggregate	638,137			25	3,190	641,352
GROUP BY	438,919			25		438,944
UNION	13,801				65	13,866

- 74% Select | 71% of INSERT/UPDATE statements are UPSERTS
- ~10% Select has Order By | Unions seldom used
- Deletes are complex (Cache Invalidation: Invalidating the offline cache data as soon as it connects to internet?)

Reference: http://odin.cse.buffalo.edu/wp-content/uploads/2015/06/2015-TPCTC-SQLite-submitted.pdf

Observations from 'Pocket Data: The Need for TPC-MOBILE' paper

Client App	Statements Executed
Google Play services	14,813,949
Media Storage	13,592,982
Gmail	2,259,907
Google+	2,040,793
Facebook	1,272,779
Hangouts	974,349
Messenger	676,993
Calendar Storage	530,535
User Dictionary	252,650
Android System	237,154

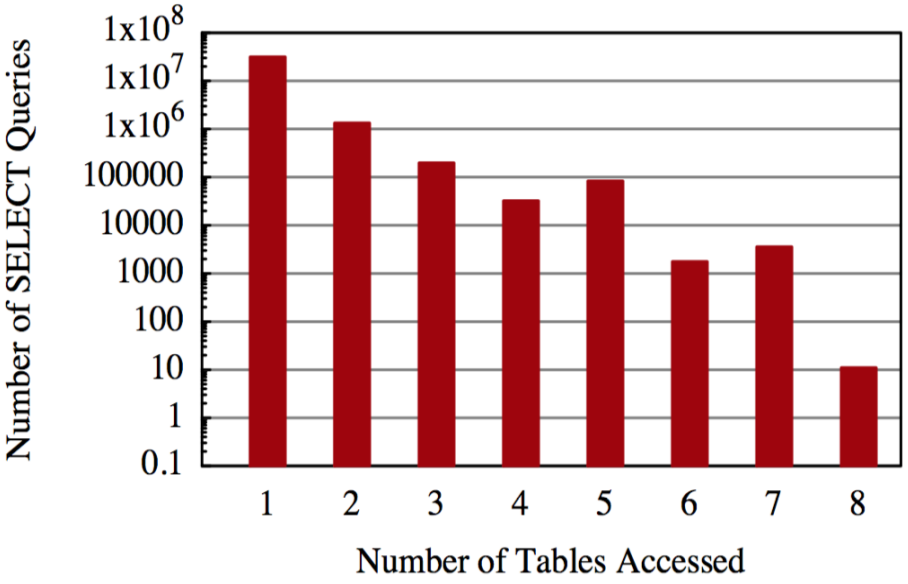
(a)

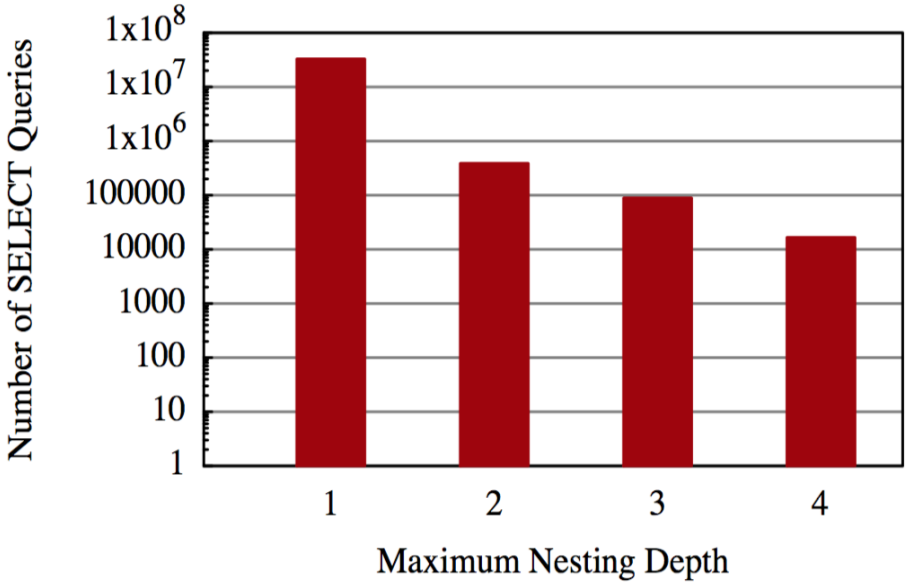
33% queries from a single service

63% queries summing up top two services

Reference: http://odin.cse.buffalo.edu/wp-content/uploads/2015/06/2015-TPCTC-SQLite-submitted.pdf

Observations from ‘Pocket Data: The Need for TPC-MOBILE’ paper





- 86% of all queries are simple single table scans/look-ups.
- Extreme – ‘Google Play Services’ queries accessing 8 distinct tables.

Reference: http://odin.cse.buffalo.edu/wp-content/uploads/2015/06/2015-TPCTC-SQLite-submitted.pdf

Observations from 'Pocket Data: The Need for TPC-MOBILE' paper

	Join Width					
Where Clauses	1	2	3	4	6	**Total**
0	1,085,154					**1,085,154**
1	26,932,632	9,105				**26,941,737**
2	1,806,843	279,811	5,970			**2,092,624**
3	384,406	80,183	29,101	1		**493,691**
4	115,107	70,891	10,696	939		**197,633**
5	28,347	15,061	1,162	17	11	**44,598**
6	212	524	591	471	3	**1,801**
7	349	22,574	333	1,048	8	**24,312**
8	35	18			6	**59**
9		541	2,564	4		**3,109**
10	159					**159**
11	545					**545**
Total	**30,353,789**	**478,708**	**50,417**	**2,480**	**28**	**30,885,422**

Reference: http://odin.cse.buffalo.edu/wp-content/uploads/2015/06/2015-TPCTC-SQLite-submitted.pdf

Observations from 'Pocket Data: The Need for TPC-MOBILE' paper

Expression Type	Expression Form	Count
Exact Lookups	`Const = Expr`	30,974,814
Other Equality	`Expr = Expr`	1,621,556
Membership Test	`Expr [NOT] IN (List or Query)`	1,041,611
Inequality on 1 constant	`Const` θ `Expr`	677,259
Disjunction	`[NOT] Expr` $\vee$ `Expr`	631,404
Bitwise AND	`Expr & Expr`	480,921
Other Inequality	`Expr` θ `Expr`	442,164
Boolean Column Cast	`[NOT] Column`	302,014
No-op Clause	`Const or (Const = Const)`	229,247
Patterned String Lookup	`Expr [NOT] LIKE Pattern`	156,309
Validity Test	`Expr IS [NOT] NULL`	87,873
Functional If-Then-Else	`CASE WHEN ...`	2,428
Range Test	`Expr BETWEEN Const AND Const`	2,393
Function Call	`Function(Expr)`	1,965
Subquery Membership	`[NOT] EXISTS (Query)`	1,584

WHERE clause expression structures, and the number of SELECT queries in which the structure appears as a conjunctive clause.

Reference: http://odin.cse.buffalo.edu/wp-content/uploads/2015/06/2015-TPCTC-SQLite-submitted.pdf

Observations from ‘Pocket Data: The Need for TPC-MOBILE’ paper

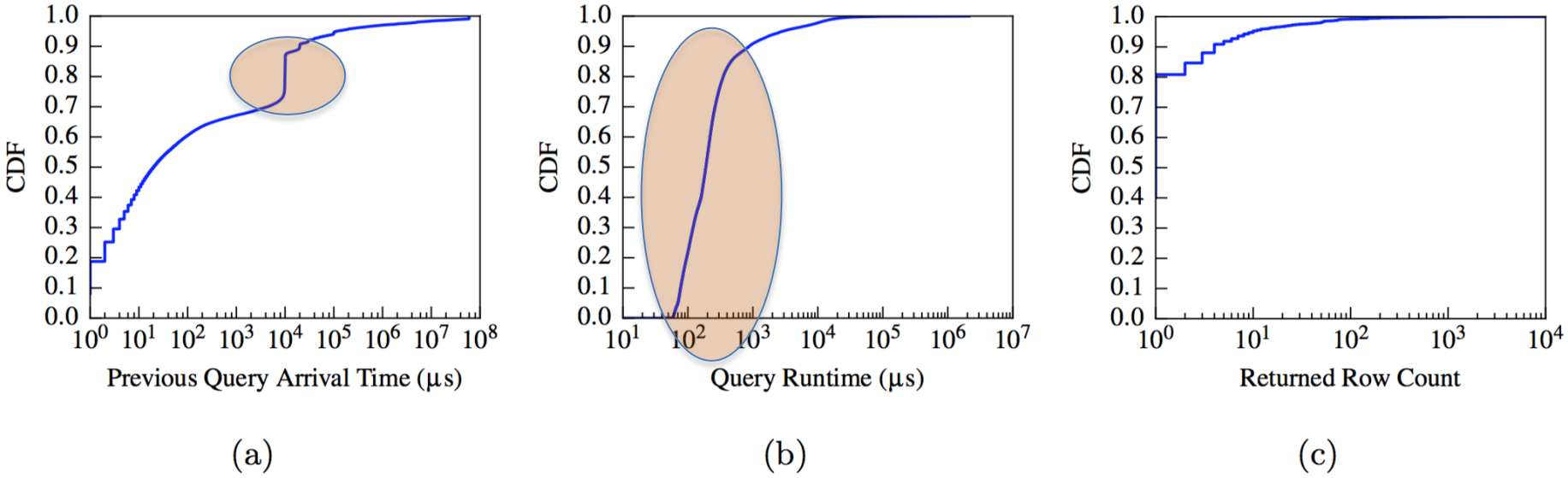


Fig. 12: **Summary Statistics for Android SQLite Queries. Distributions of (a) inter-query arrival times, (b) query runtimes, and (c) rows returned per query.**

- 20% queries periodic (File Locks?)
- 85% queries run in 0.1ms
- 80% queries returned single row (key-value lookup)

Reference: http://odin.cse.buffalo.edu/wp-content/uploads/2015/06/2015-TPCTC-SQLite-submitted.pdf

Ideation

- Analysis per Application (Read % , Read/Write ratio)
  - Long tail distribution skews results.
- Cluster based analysis
  - Analyze patterns within cluster
  - Generalize the behavior
  - Explain the behavior
  - How certain that a new app of this cluster will behave same?

Ideation

- Cluster Analysis [Contd..]
  - Finding similar clusters for each feature. Combine them into one if they behave same.
  - Split a cluster into two if there are two sets of query access patterns and they can be explained.
  - Frequency of app usage within cluster should not demand different benchmarks.
    - It should be driven by scale factor and burst factors?

Steps ahead

- Identifying and finalizing the right features
- Phone data log file extraction.
- Implement the ideas.
  - Per app basis analysis & cluster based analysis

Lightweight Runtimes

Team Sparkle

Dhinesh
Shiva
Keno
Guru

Project Objectives

Re-design a database to support the Internet of Things
Study capabilities of currently available hardware
Understand streaming data characteristics

Intel Galileo Board

Next Steps

Study behaviour under memory pressure
Java and GC effects on Galileo
Characterize specific workloads we want to support
    rapid inserts
    rapid queries
    range queries
Find bottlenecks in current implementations
Benchmark streaming data eg. light sensor from phone

Study behaviour under memory pressure

Memory and Performance are related.
A smaller Java memory heap - **256 MB**

Memory heap fills lot quicker

Is garbage collected more frequently, so longer run time.

May lead to out-of-memory errors.

Heap Size?

Dataset size : 10mb
Set min heap size : *Java -Xmx100m Main*
Do a binary search on the binary heap size, to find the exact minimum amount of heap memory needed for the program to run.

Heap Size (cont...)

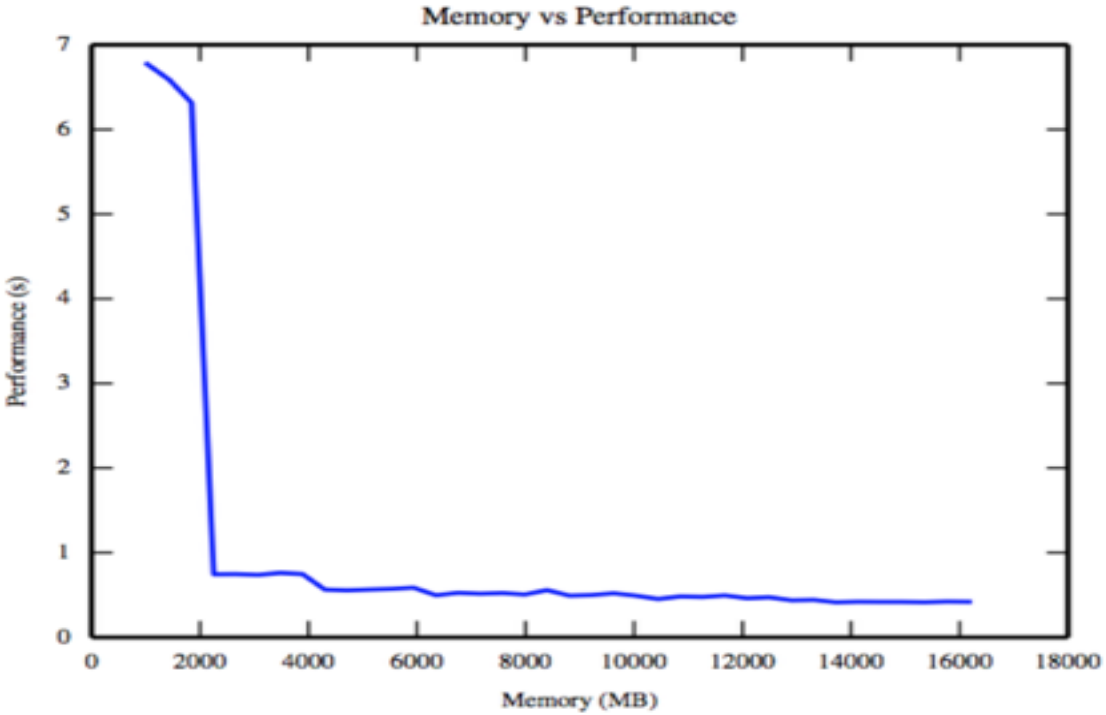
What is max heap allocated for the program?

- The threshold of allocated heap after which additional memory does not yield any (significant) performance gain

How did we find max heap?

- Increase min heap by multiplier (0.4x) every iteration
- Stop when performance is within 3% over 5 (consecutive) iterations

Runtime vs Heap Size -Host Machine



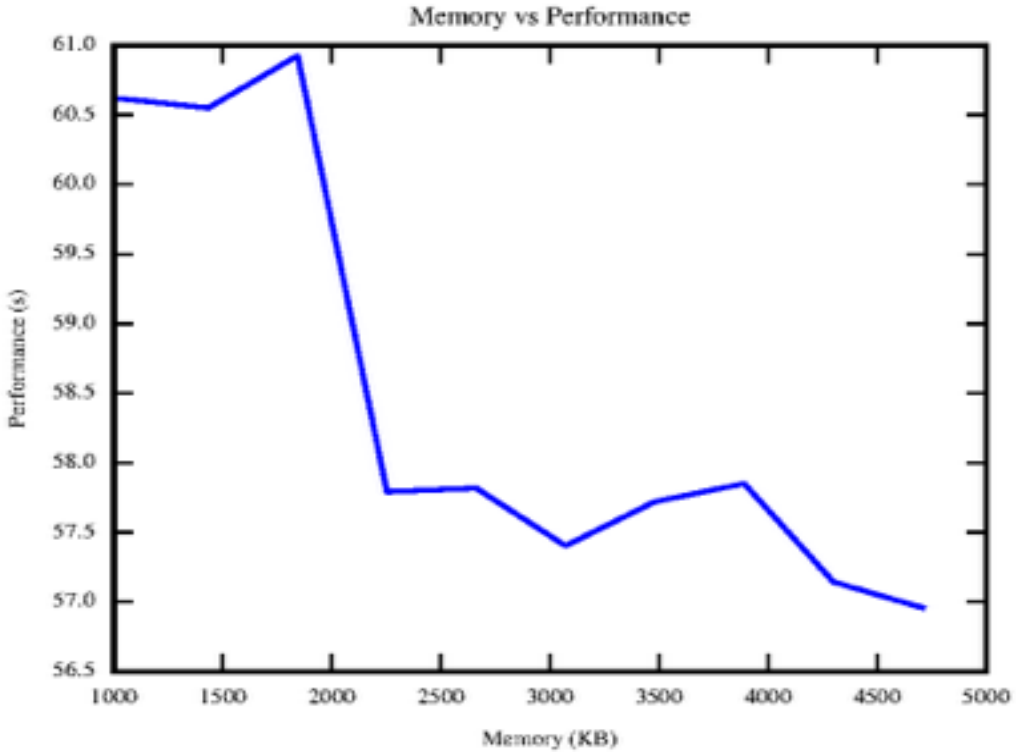
KB

TPCH-1 Query

Environment:

8gb RAM
i7 processor

RunTime vs Heap Size - Galileo board



TPCH-1

Environment:
-256 mb ram
-Intel Quark Processor

Runtime vs Heap Size TPC-H 3

Results are non deterministic for min heap!
Java -Xmx49m Main does not always work for tpch3
why?? java.lang.OutOfMemoryError

Next Steps

Study behaviour under memory pressure
Java and GC effects on Galileo
To do : Design decisions based on the findings
Characterize specific workloads we want to support
    rapid inserts
    rapid queries
    range queries
Find bottlenecks in current implementations
Benchmark streaming data eg. light sensor from phone

LLVM Query Runtime

VALKyrie

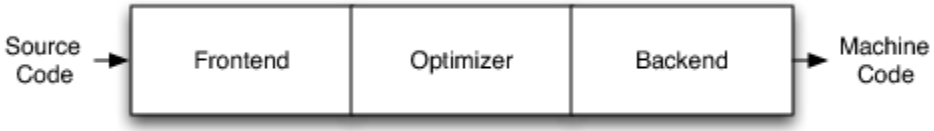
Arindam
Kaushik

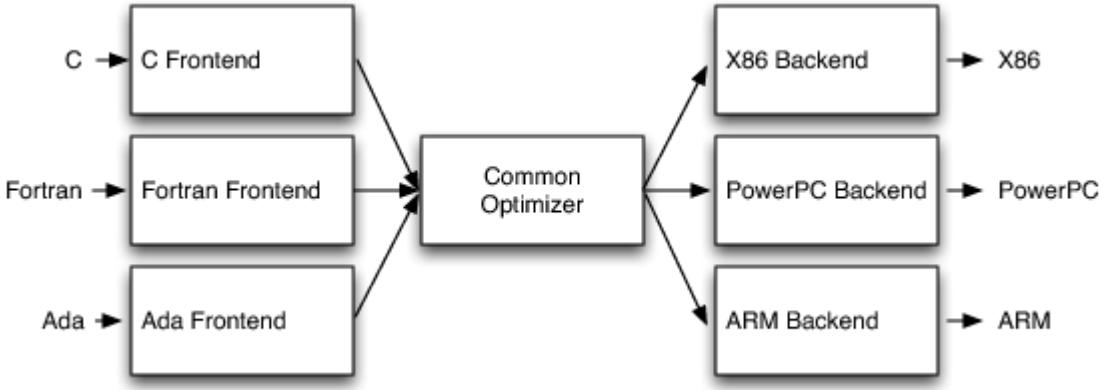
Ladan
Vinayak

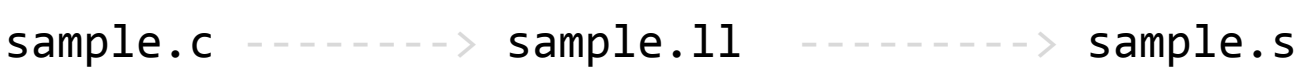
Progress

- LLVM Intermediate Representation (IR)
- Ways of generating IR
- Benchmark setup and initial experiments

LLVM Intermediate Representation (IR)







sample.c

```
> clang -Os -S -emit-llvm sample.c -o sample.ll
```

sample.ll (IR)

```
> opt-3.0 -S sample.ll
```

sample.ll (IR Optimized)

```
> llc-3.0 -O3 sample.ll -march=x86-64 -o sample.s
> llc-3.0 -O3 sample.ll -march=x86 -o sample.s
> llc-3.0 -O3 sample.ll -march=arm -o sample.s
```

sample.s (machine code)

```
> lli sample.ll
```

Hello World! (Output)

LLVM IR

```
#include<stdio.h> int main() { printf(“Hello World!\n”); }
```

```
> clang -emit-llvm -S helloworld.c -o helloworld.ll
```

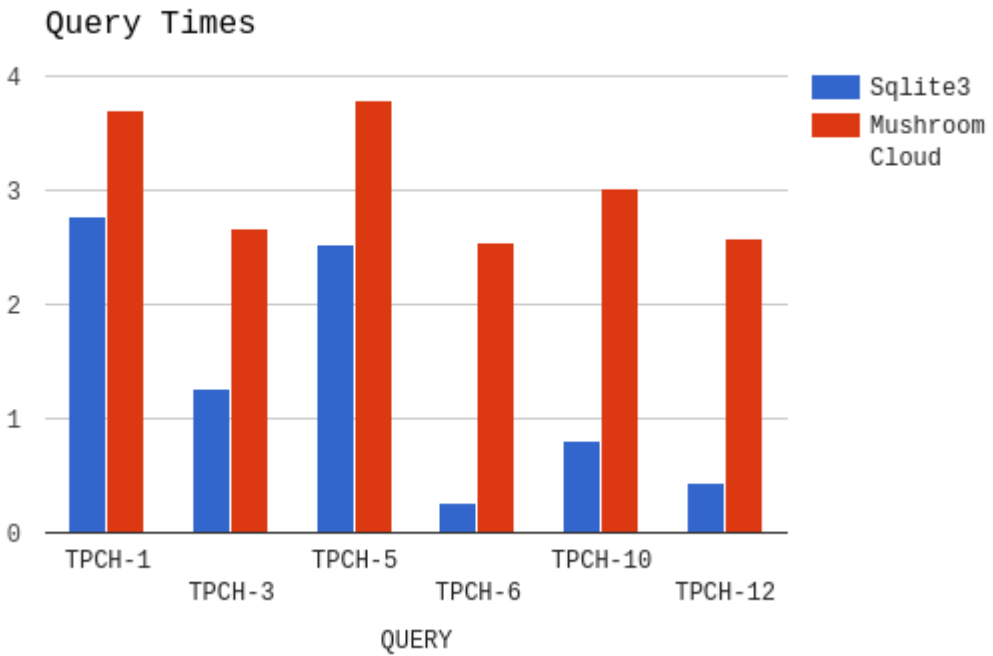
```
@.str = private unnamed_addr constant [14 x i8] c"Hello World!\0A\00"

define i32 @main() {
  %1 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([14 x i8]* @.str, i32 0, i32
0))
  ret i32 0
}
declare i32 @printf(i8*, ...)
```

```
> lli helloworld.ll
Hello World!
```

Ways to generate IR

- Java bindings
  - robovm
  - Java Library for creating LLVM IR
- Manual Translation
- Python bindings [llvmlite]
- Official C++ API

Baseline Benchmark



Scale Factor 0.1
i3 3120M @ 2.5 GHz
4 GB RAM
Average of 3
Warm cache

Plan of action

- In memory benchmarks
- Implement IR generation prototypes