MBWU: Benefit Quantification for Data Access Function Offloading

Jianshen Liu [1], Philip Kufeldt [2], Carlos Maltzahn [1]

[1] University of California, Santa Cruz, [2] Seagate Technology

June 20, 2019

This project is funded in part by NSF OAC-1836650, CNS-1764102, CNS-1705021, and the Center for Research in Open Source Software (cross.ucsc.edu).

1

Is It Worth to Offload?

Active Disks: Programming Model, Algorithms and Evaluation

Anurag Acharya
Dept. of Computer Science
University of California
Santa Barbara

Mustafa Uysal
Dept. of Computer Science
University of Maryland
College Park

Joel Saltz
Dept. of Computer Science
University of Maryland
College Park

A Case for Intelligent Disks (IDISKs)

Kimberly Keeton, David A. Patterson and Joseph M. Hellerstein

Computer Science Division
University of California at Berkeley
387 Soda Hall #1776
Berkeley, CA 94720-1776
{kkeeton, patterson, jmh}@cs.berkeley.edu

Active Disk Meets Flash: A Case for Intelligent SSDs

Sangyeun Cho (University of Pittsburgh), Chanik Park (Samsung Electronics Co., Ltd.), Hyunok Oh (Hanyang Univesity), Sungchan Kim (Chonbuk National University), Youngmin Yi (University of Seoul) and Gregory R. Ganger (Carnegie Mellon University)

CMU-PDL-11-115
November 2011

Fast, Energy Efficient Scan inside Flash Memory SSDs

Sungchan Kim
Chonbuk National University, Korea
sungchan.kim@chonbuk.ac.kr

Hyunok Oh
Hanyang University, Korea
hyunok.oh@hanyang.ac.kr

Chanik Park
Samsung Electronics Co., Ltd., Korea
ci.park@samsung.com

SSD In-Storage Computing for List Intersection

Jianguo Wang[†] Dongchul Park[§] Yang-Suk Kee[§]
Yannis Papakonstantinou[†] Steven Swanson[†]

YourSQL: A High-Performance Database System Leveraging In-Storage Computing

Insoon Jo, Duck-Ho Bae, Andre S. Yoon, Jeong-Uk Kang, Sangyeun Cho, Daniel DG Lee, Jaeheon Jeong
Memory Business, Samsung Electronics Co.

ABSTRACT

This paper presents *YourSQL*, a database system that accelerates data-intensive queries with the help of additional in-storage computing capabilities. YourSQL realizes very early filtering of data by offloading data scanning of a query to user-programmable solid-state drives. We implement our system on a recent branch of MariaDB (a variant of MySQL). In order to quantify the performance gains of YourSQL, we evaluate SQL queries with varying complexities. Our result shows that YourSQL reduces the execution time of the whole TPC-H queries by 3.6×, compared to a vanilla system. Moreover, the average speed-up of the five TPC-H queries with the largest performance gains reaches over 15×. Thanks to this significant reduction of execution time, we observe sizable energy savings. Our study demonstrates that the YourSQL approach, combining the power of early filtering with end-to-end datapath optimization, can accelerate large-scale analytic queries with lower energy consumption.

1. INTRODUCTION

Delivering end results quickly for data-intensive queries is key to successful data warehousing, business intelligence, and analytics applications. An intuitive (and efficient) way to speed them up is to reduce the volume of data being transferred across storage network to a host system. This can be achieved by either filtering out extraneous data or transferring intermediate and/or final computation results [18].

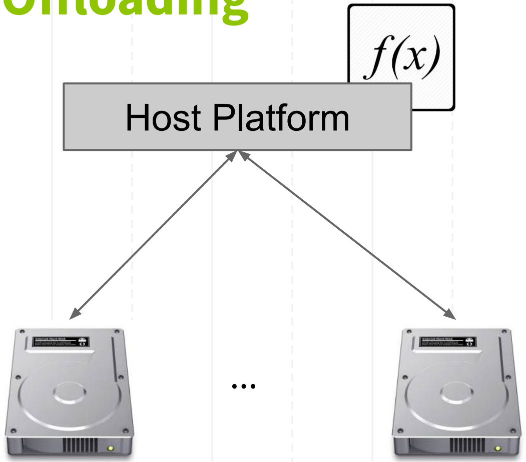
The former approach is called *early filtering*, a typical ex-

fect solution, either. Leaving the expense aside, the amount of data transferred from storage devices remains unchanged because the data must be transferred in any case to filter servers or FPGAs before being filtered.

Contrary to the prior work, we argue that early filtering may well take place at the earliest point possible—within a storage device. Besides a fundamental computer science principle—when operating on large datasets, do not move data from disk unless absolutely necessary—, modern solid-state drives (SSDs) are not a dumb storage any longer [7, 9, 11, 13, 16, 17, 20]. Therefore, we have explored a novel database system architecture where SSDs offer compute capabilities to realize faster query responses. Some prior work aims to quantify the benefits of in-storage query processing. Do et al. [11] built a "smart SSD" prototype, where SSDs are in charge of the whole query processing. Even though this work lays the basis for in-storage query processing, there remain large areas for further research due to its limitations. First, it focuses on proving the concept of SSD-based query processing but pays little attention to realizing a realistic database system architecture. Not only would join queries be unsupported, but internal representation and layout of data must be converted to achieve reasonable speed-up. Moreover, the hardware targeted by this work (i.e., SATA/SAS SSDs) is outdated and the corresponding results may not hold for future systems. Indeed, its performance advantages mainly result from the higher internal bandwidth inside an SSD compared to the external SSD bandwidth limited by a typical host interface (like

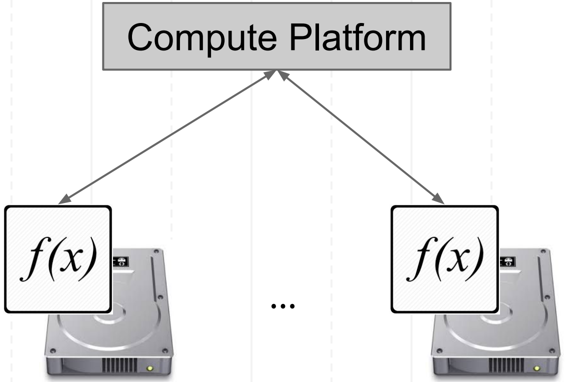
Cost/Benefit of Offloading

Offloading



Cost/Benefit of Offloading

Offloading



Embedded Platform

Possible Benefits

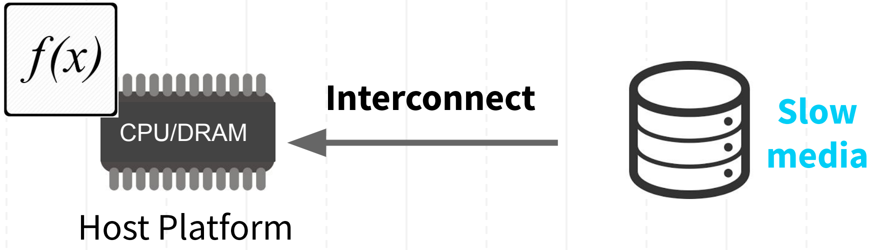
- Data Translation Reduction
- Data Transmission Size Reduction
- Software Layer Reduction
- Power Consumption Reduction
- Application Performance Increment
- Resource Utilization Increment
- ….

⬆️ storage device cost $\not\Rightarrow$ ⬆️ overall platform cost

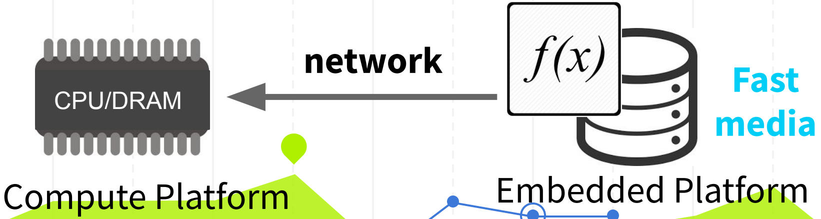
Placement of Data Access Functions

Different storage media, different workloads $\Longrightarrow$ different cost-optimal placements of functions

Move *data access* function close to DRAM to hide latency



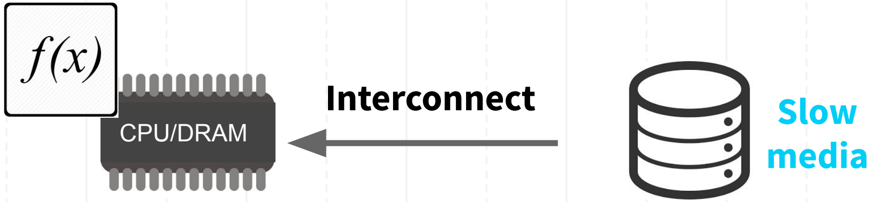
Move *data access* function close to data to save bandwidth cost



Placement of Data Access Functions

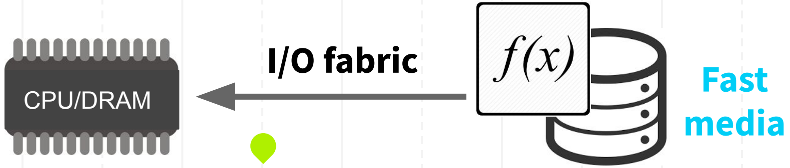
Different workloads, different storage media $\Longrightarrow$ different cost-optimal placements of functions

Move *data access* function close to DRAM to hide latency



Host Platform

Move *data access* function close to data to save bandwidth cost



Compute Platform

Embedded Platform

$f(x)$ ***Data access function***

Examples:

- GET/PUT in K/V Store
- read/write in File System
- SELECT/PROJECT in DBMS
- H5Sselect in HDF5

Workload:

- data access function calls

Throughput:

- data access function calls per second (aka ops/sec, IOPS, OPS)

Problem: How to quantify cost/benefit?

2 Measurement Methodology

Efficiency Comparison for Different Platforms

Different storage media, different workloads $\Longrightarrow$ different cost-optimal placements of functions

We need a normalization that is

- **Platform-independent**
  Reference point across host and embedded platforms
  Based on amount of work measured in workload operations (as opposed to CPU cycles)

Efficiency Comparison for Different Platforms

Different workloads, different storage media $\Longrightarrow$ different cost-optimal placements of functions

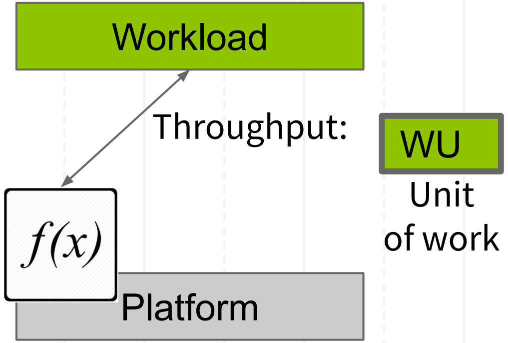
We need a normalization that is

- **Platform-independent**
  Reference point across host and embedded platforms
  Based on amount of work measured in workload operations (as opposed to CPU cycles)

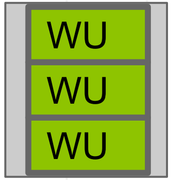
- **Workload-dependent**
  Workload operations are implemented by the data access function under study
  Examples: GET/PUT K/V ops, read/write FS ops, db transactions

- **Media-dependent**
  Cost-optimal placement of data access function sensitive to types of storage media
  Examples: Spinning media (slow), flash media (fast)

Efficiency Normalized by Work Performed



Platform A : 8 units of work

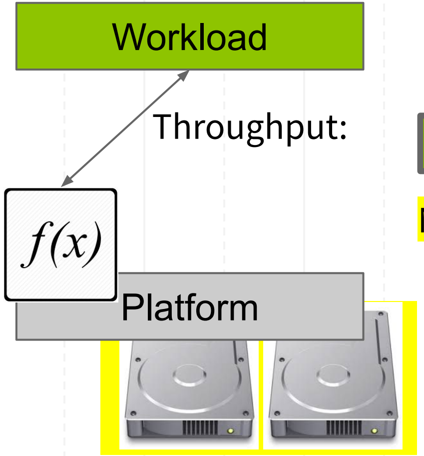


Platform B : 3 units of work

Platform Efficiency

Cost efficiency	\$/WU
Power efficiency	kWh/WU
Space efficiency	m^3/WU

Efficiency Normalized by Work Performed Limited by Media



Media-based unit of work

Platform A:
8 meda-based units of work

Platform B:
3 media-based units of work

Platform Efficiency

Cost efficiency	\$/MBWU
Power efficiency	kWh/MBWU
Space efficiency	m^3/MBWU

How to Construct a MBWU(workload, media)

Construct a MBWU

1. Run *workload* on platform that is only limited by *storage media*, with all external caching effects eliminated/disabled
2. Determine maximum steady-state *throughput*
3. 1 MBWU ← that *throughput*

- MBWU *construction* is fully repeatable
- Intended for all workloads, storage media
- **Not**: online method during production workloads

Measure MBWUs of a platform

1. Run *workload* on platform under study
2. Determine maximum steady-state *throughput* of platform under study using the same *workload*
3. Divide *throughput* by constructed MBWU

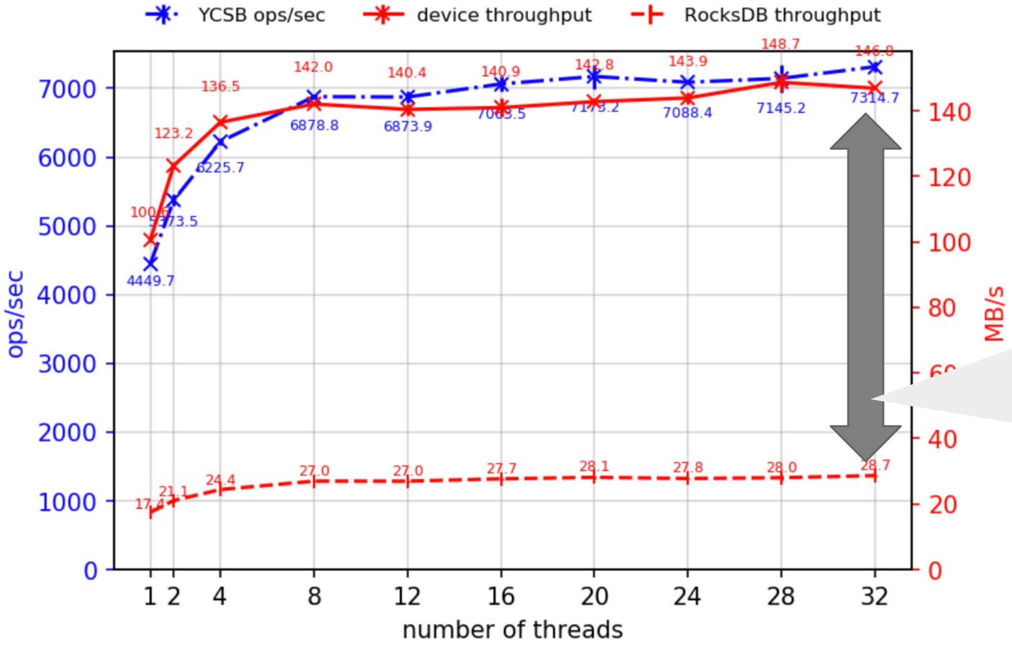
Compare platforms

1. Measure MBWUs for each platform
2. Determine $, kWh (under *workload*), volume of each platform
3. Normalize by MBWU: $/MBWU, kWh/MBWU, m^3/MBWU

3 Example Evaluation

The Workload

Key-value data management as an example workload to be offloaded.



- **RocksDB** as the key-value store engine
- **YCSB** as the workload generator

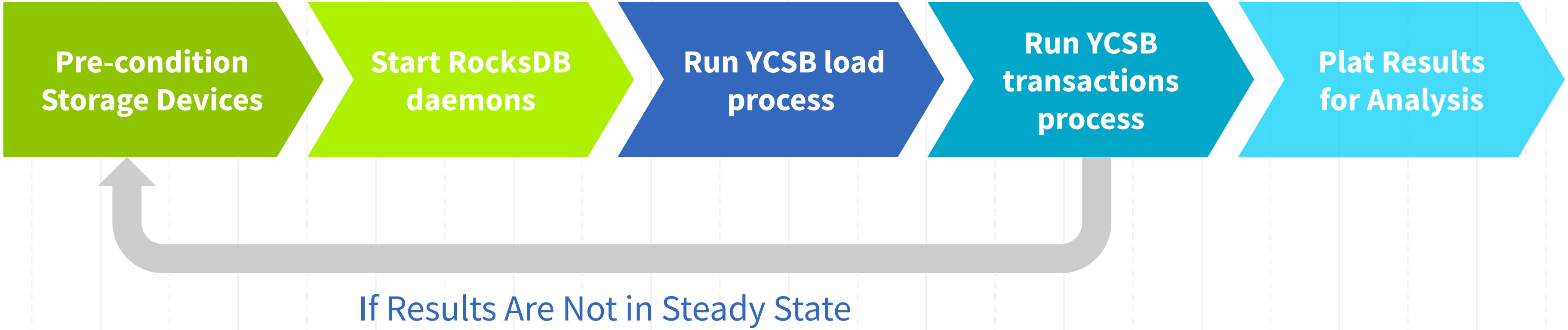
Why this workload?

6x traffic amplification

The Workload

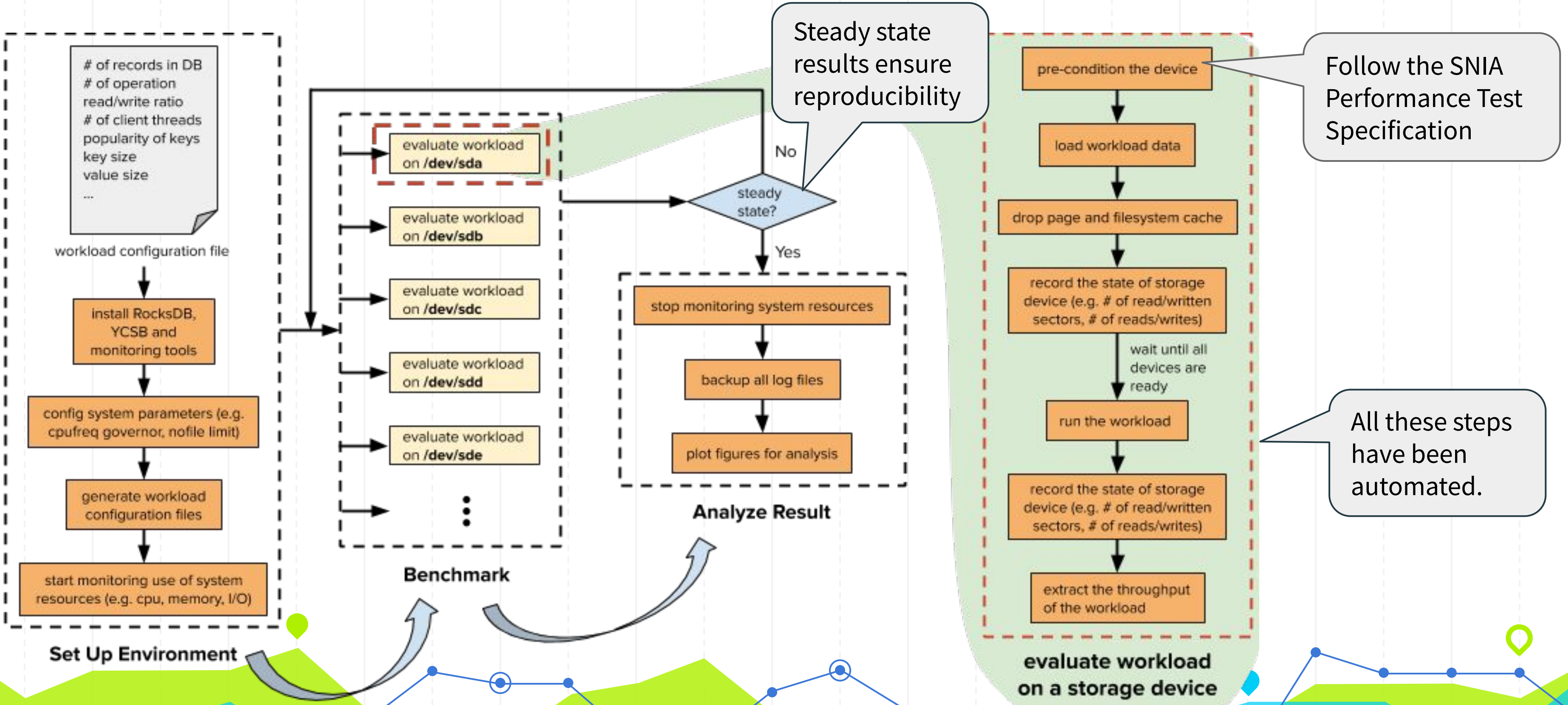
- Key-value data management is a typical high-selectivity data access function.
- 6x amplification means more than 5x extra expense on the I/O fabric to support the bandwidth that is not directly relevant to user applications.
- There is nothing to prevent the MBWU-based measurement methodology from being applied to other workloads, such as database operations workload.

Evaluation Process

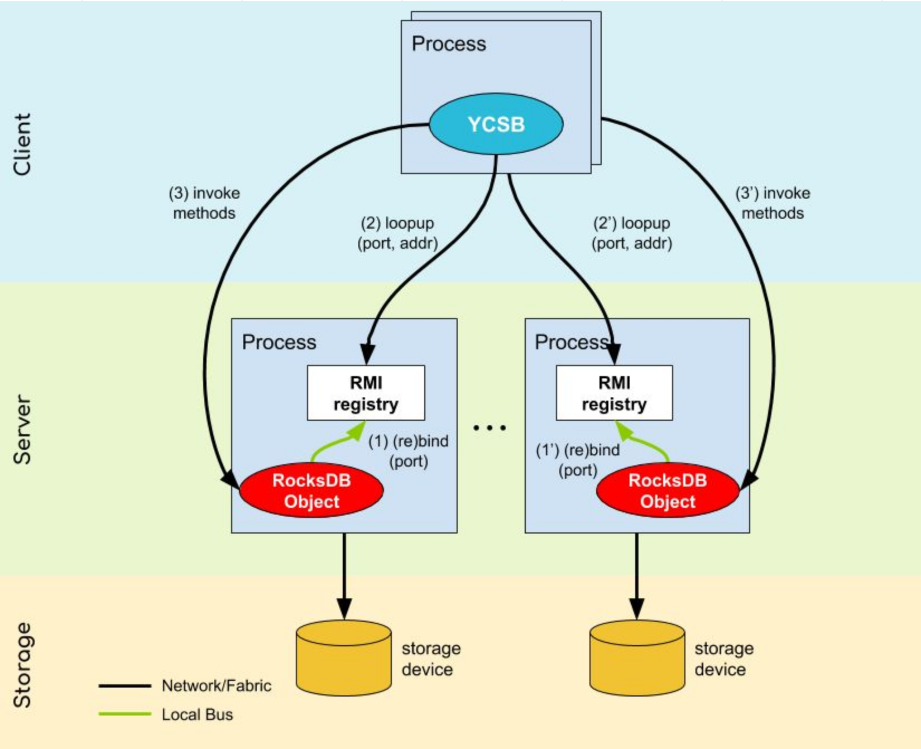


Monitor and record utilization of CPU, memory, device I/O, network, and power for the platform during the whole evaluation process.

Repeatable Evaluation Process



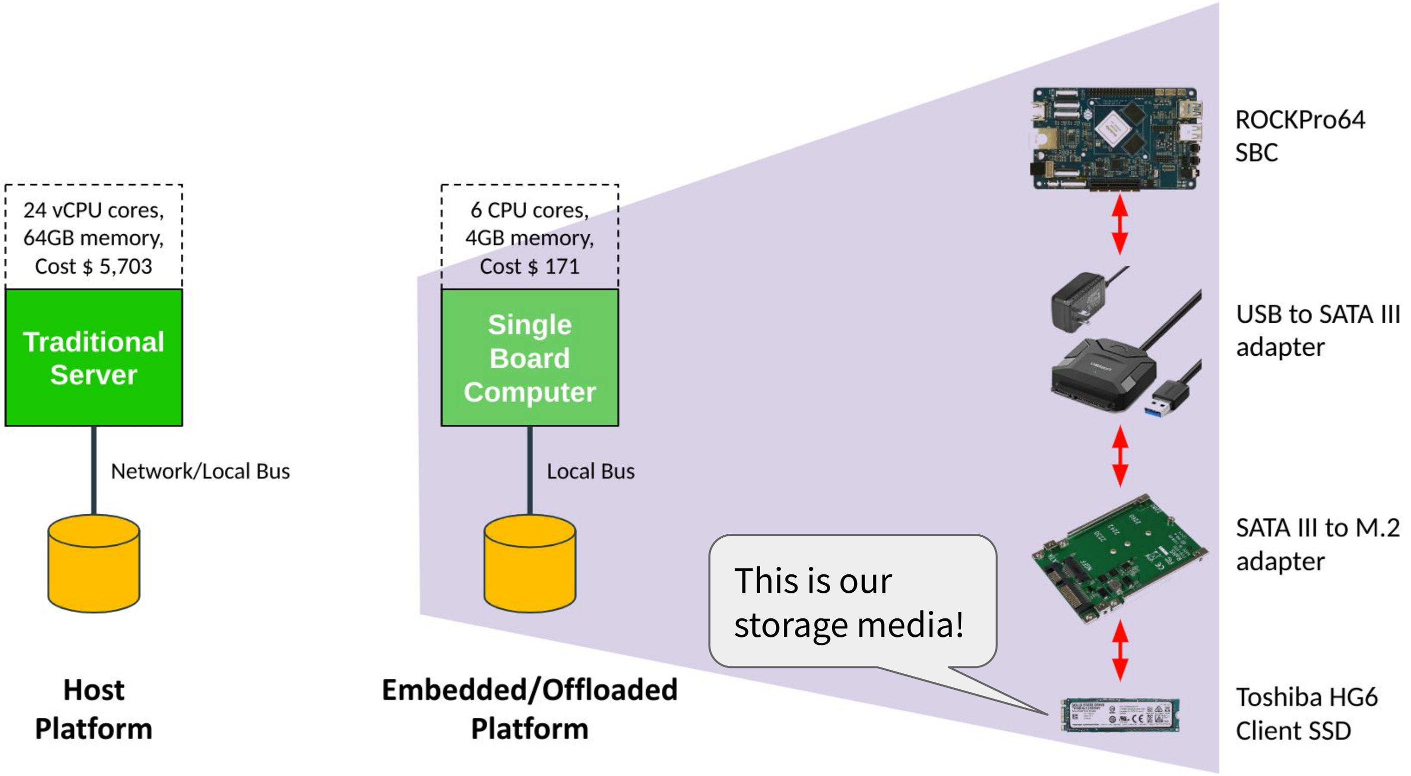
RocksDB RMI Server



- It exposes all public interfaces (e.g., *open()*, *close()*, *get()*, *put()*, *delete()* of a RocksDB object to network securely by binding this object to an RMI registry.
- A YCSB process looks up the corresponding RocksDB object from a specified RMI registry.
- YCSB passes down I/O operations to the exposed RocksDB insterfaces.

4 Prototype Evaluation

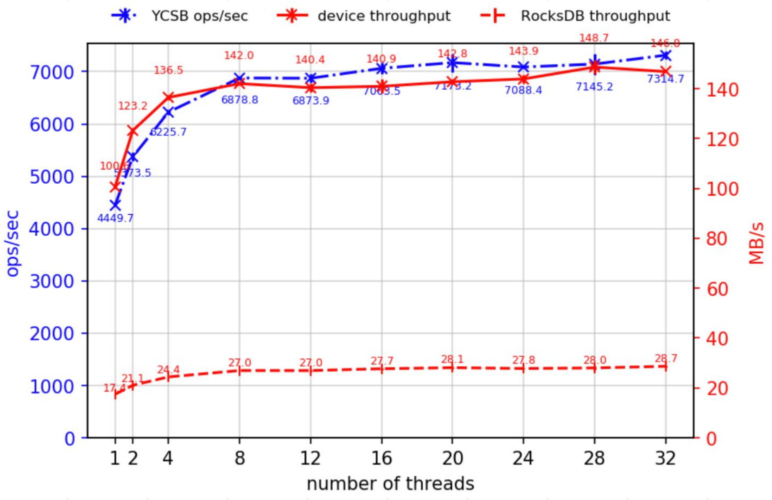
Infrastructure Setup



The Key-value Workload in Experiment

- The key size is 16 bytes, and the value size is 4 KiB.
- The read/write ratio is 50/50 following a Zipf distribution for data accessing.
- The total size of dataset is 40 GiB.

The Value of An MBWU

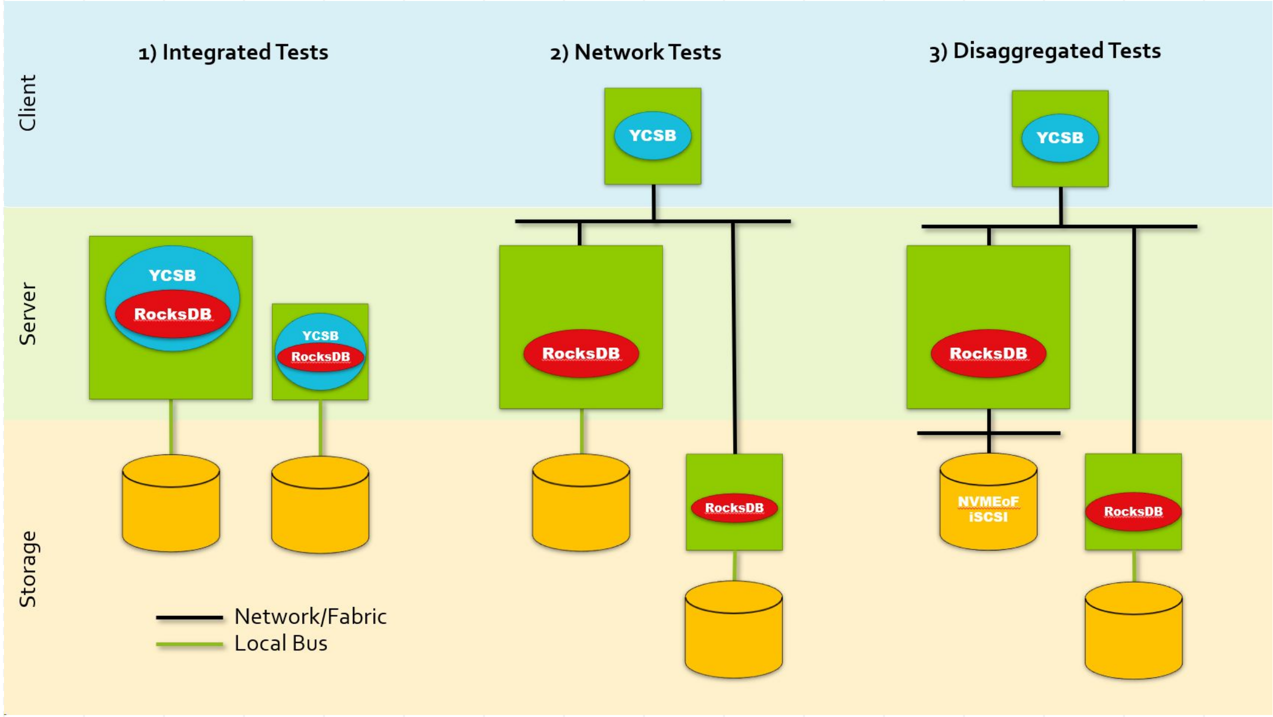


By running the evaluation prototype on our host platform, we got the value of a single MBWU for this workload:

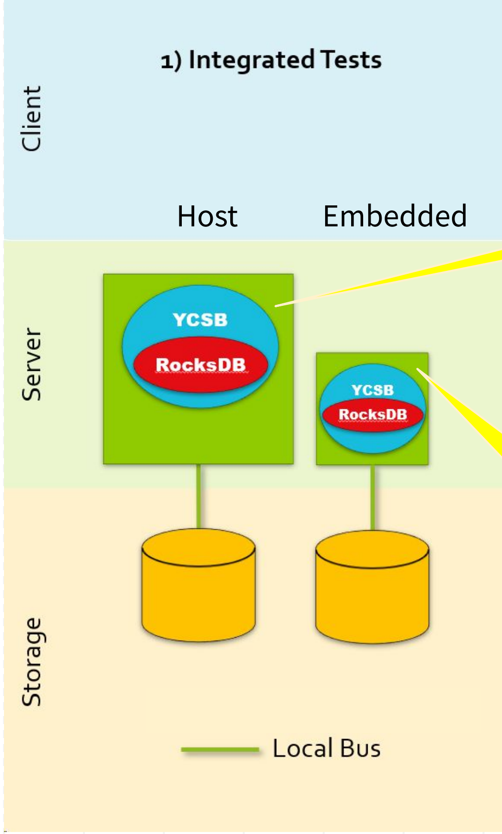
1 MBWU = 7314.6 ops/sec

Now, we can evaluate efficiency of different platforms!

Three-stage Test Setup



- **Integrated Tests**
  Evaluate the benefits of leveraging cost-efficient hardware to provide key-value data store.
- **Network Tests**
  Evaluate how the introduction of the front-end network affects the benefit results.
- **Disaggregated Tests**
  Evaluate the benefits of removing the back-end network requirement for data management traffic.



Results From Integrated Tests

Our **host platform** can generate 6 MWBUs.

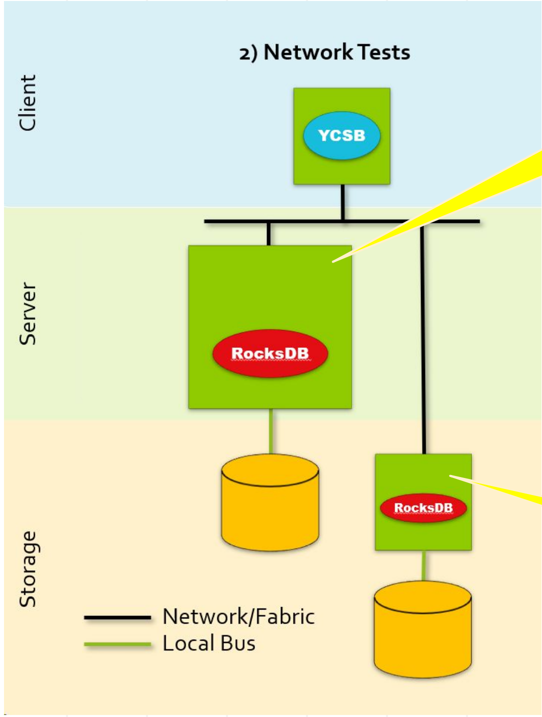
Offload

64% less $/MBWU
40% less kWh/MBWU

Our **embedded platform** can generate 0.5 MBWUs.

- **Integrated Tests**
  Evaluate the benefits of leveraging cost-efficient hardware to provide key-value data store.
- **Network Tests**
  Evaluate how the introduction of the front-end network affects the benefit results.
- **Disaggregated Tests**
  Evaluate the benefits of removing the back-end network requirement for data management traffic.

Results From Network Tests



Our **host platform** can generate 5.2 MWBUs.

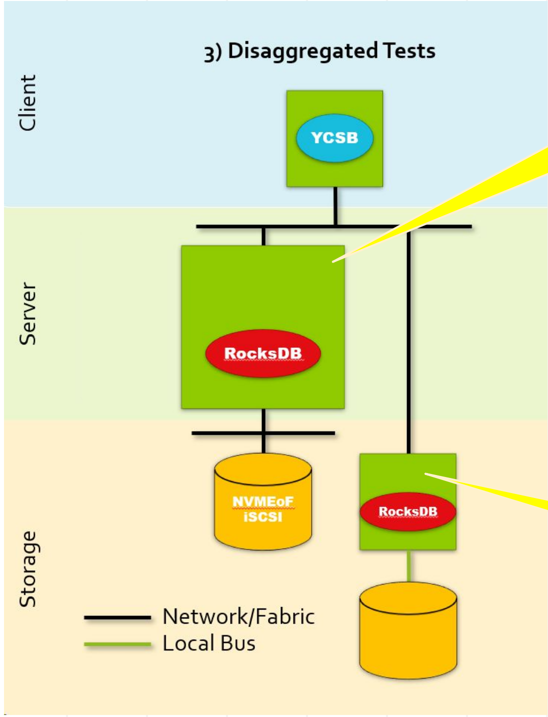
Offload

58% less $/MBWU
46% less kWh/MBWU

Our **embedded platform** can generate 0.37 MBWUs.

- **Integrated Tests**
  Evaluate the benefits of leveraging cost-efficient hardware to provide key-value data store.
- **Network Tests**
  Evaluate how the introduction of the front-end network affects the benefit results.
- **Disaggregated Tests**
  Evaluate the benefits of removing the back-end network requirement for data management traffic.

Results From Disaggregated Tests



Our **host platform** can generate 3.2 MWBUs.

Offload

74% less $/MBWU

Our **embedded platform** can generate 0.37 MBWUs.

- **Integrated Tests**
  Evaluate the benefits of leveraging cost-efficient hardware to provide key-value data store.
- **Network Tests**
  Evaluate how the introduction of the front-end network affects the benefit results.
- **Disaggregated Tests**
  Evaluate the benefits of removing the back-end network requirement for data management traffic.

Conclusion

The MBWU Measurement Methodology

- provides an instruction to answer the following question:
  - How efficient is a platform to run a *given workload* over a *specific storage media*?
- apple-to-apple efficiency comparisons for different platforms.
- benefits quantification for functions offloading from traditional host platforms to embedded platforms.

Conclusion

The Evaluation Prototype

- automates the evaluation process for quantifying benefits of offloading customized key-value workloads.

Target users: storage device/systems designers

THANKS!

Any questions?

Carlos Maltzahn

carlosm@ucsc.edu

Cross.ucsc.edu (Eusocial Storage Devices)