# MilliSort and MilliQuery: Large-Scale Data-Intensive Computing in Milliseconds

**Yilong Li***, Seo Jin Park*, John Ousterhout

Stanford University

PLATFORMLAB

*co-first authors

# Introduction

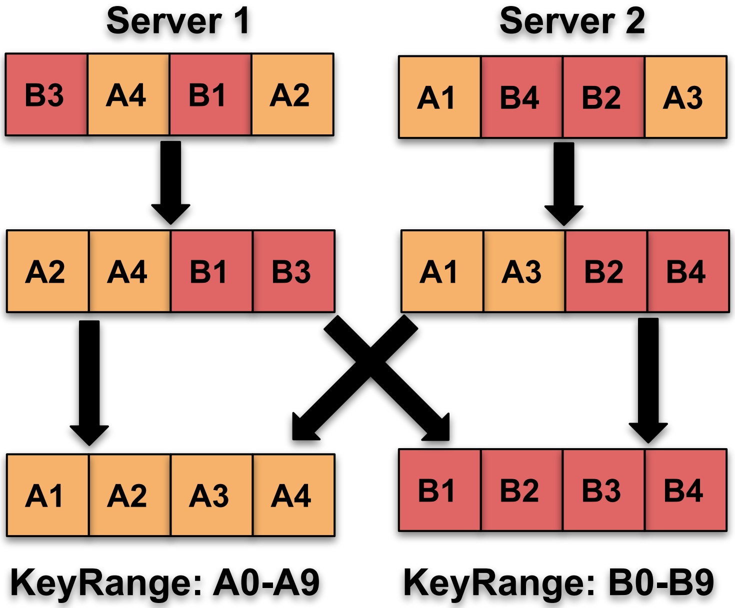
- **Current datacenter applications couple scale and time**
  - Batch processing applications
    - Scale to clusters with 1000s of nodes
    - Execute for long periods of time: minutes to hours
  - Serverless computing: short-lived tasks, small Lambda functions (1-2 vCPUs)
- **Flash burst: large-scale computing in milliseconds**
  - Harness hundreds or thousands of servers
  - Very short lifetime (e.g., 1-10 ms)
  - Enable data-intensive real-time analytics
- **Goal: understanding the limits of flash bursts**
  - What is the smallest possible timescale to operate efficiently?
  - What is the largest number of servers that can be harnessed?
  - What aspects of the current systems limit the duration and scale?

# Contributions

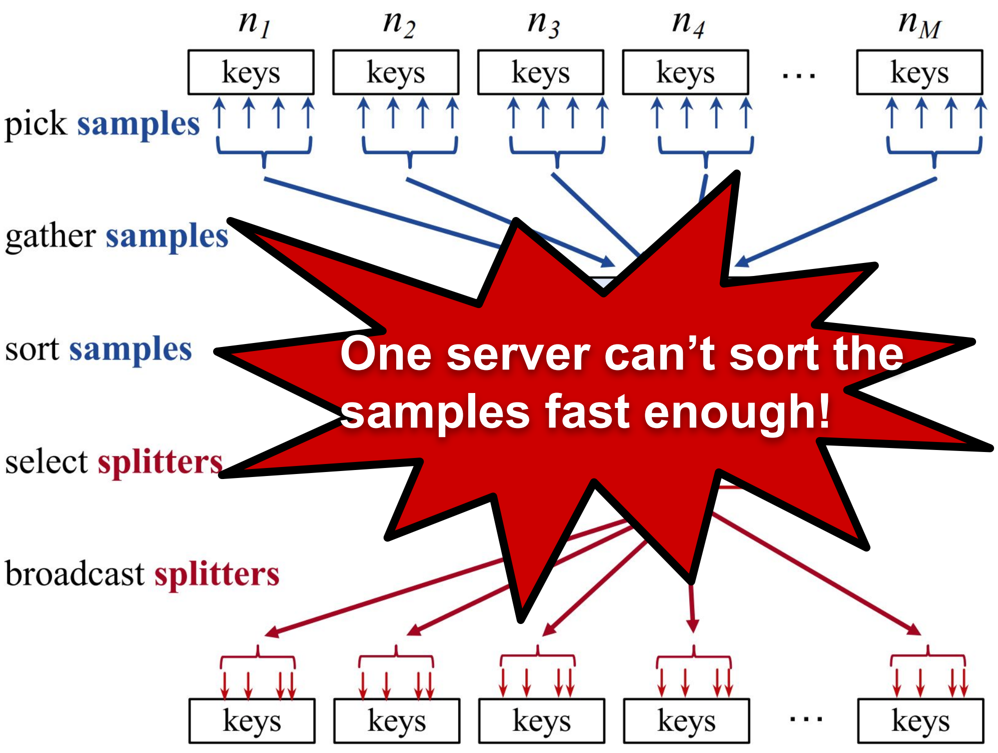
- **Developed two example apps to understand flash bursts**
  - **MilliSort**: distributed sorting of 100-byte records
  - **MilliQuery**: three representative SQL queries
  - Goal: process as much data as possible in 1 ms (or 10 ms) using unlimited resources
  - Assumption: input data already exist in memory
- **Lessons learned**
  - **Feasibility**: flash bursts can harness 100s of servers efficiently even under 1 ms
  - **Scaling**: total data processed grows at least quadratically with the time budget
  - **Limiting factors** (both can be attributed to small-message throughput)
    - Coordination overhead
    - Shuffle cost

# MilliSort Algorithm

- **Challenge of distributed sorting**
  - Complex data flow: any record may end up on any server
- **MilliSort implements a distributed bucket sort algorithm**
  - Optimize network bandwidth usage
- **Four basic steps:**
  - Local sort: each server sorts its initial data
  - Partitioning: determine the key range each server stores after the sorting (details later)
  - Shuffle data: each server transmits its records to the targets
  - Rearrangement: merge-sort incoming records as they arrive



# Challenge of Partitioning

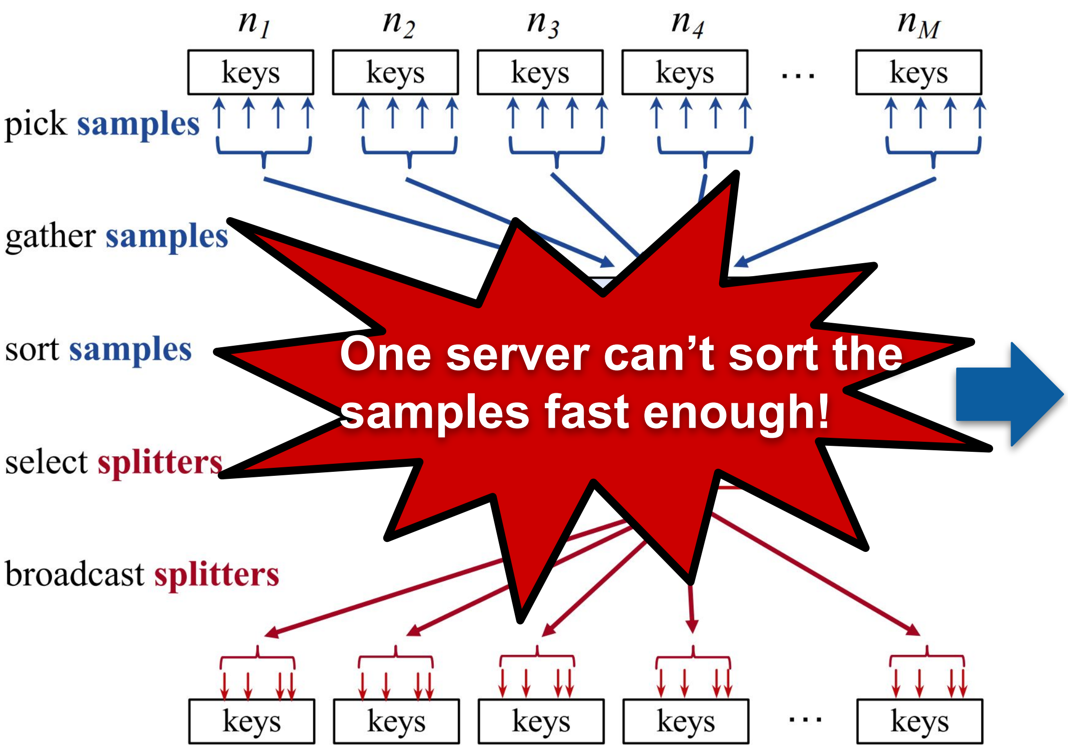



Partition by regular sampling

**Terminology:**

- **Sample**: estimate the key distribution
- **Splitter**: split key space into buckets

# Challenge of Partitioning




**Partition by regular sampling**

**Solution: recursive distributed sort**

- Select a small group of servers to sort the samples
- Apply the same distributed bucket sort algorithm

**Apply more levels of recursion for larger clusters.**

# MilliQuery

Complexity of Coordination

**Q1: embarrassingly parallel scan-aggregate query**

- Count Wikipedia article views by language

**Q2: like Q1, but repartition records by shuffle before aggregation**

- Find top 10 IP addresses by the number of edits to Wikipedia

**Q3: distributed join operation that requires multiple shuffles**

- Complex analytics on GitHub data

**MilliSort and MilliQuery capture a wide range of interesting behaviors**

# Experiment Setup

- **Hardware configuration**

| | |
|---|---|
| **CPU** | Xeon Gold 6148 (2 sockets × 20 cores @ 2.40GHz) |
| **RAM** | 384 GB DDR4-2666 |
| **Networking** | 100Gbps Intel Omni-Path Interconnect |

- **Prototype built atop RAMCloud's transport system**
  - Kernel bypass: 5 μs RTT, 25 Gbps network bandwidth
  - Message throughput limited by the single dispatch thread
- **Run four servers on each machine to better utilize the network**
  - Each server has 8 cores and 25 Gbps network bandwidth
- **We had access to 70 machines, which allowed up to 280 servers**

# Overall Performance

**MilliSort can sort 0.84M records using 120 servers in 1ms.**

| Time budget | Total records processed | | | |
|---|---|---|---|---|
| | **MilliSort** | **Q1** | **Q2** | **Q3** |
| **1 ms** | 0.84M | 47.6M* | 6.72M | 0.034M |
| **10 ms** | 26M* | 980M* | 224M* | 2.24M* |

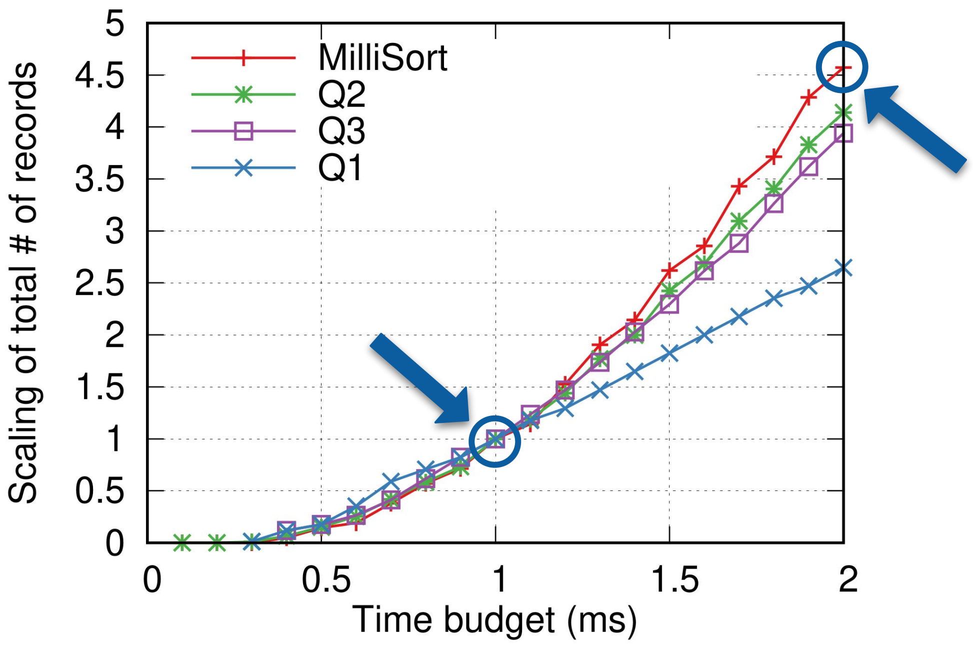
| Time budget | # servers used | | | |
|---|---|---|---|---|
| | **MilliSort** | **Q1** | **Q2** | **Q3** |
| **1 ms** | 120 | 280* | 140 | 60 |
| **10 ms** | 280* | 280* | 280* | 280* |

*limited by the cluster size in experiment

- In 1 ms, all applications except Q3 can harness **>100** servers
- In 10 ms, all applications can scale **beyond 280** servers

**Super-linear increase in total data processed?**

# Quadratic Scaling w/ Time Budget




- **Total data processed grows at least quadratically with the time budget**
  - Both #servers and #records/server grow at least linearly
  - Not a lot of work can be done for time budgets less than 1 ms

# Why not more servers?

- **Time breakdown (μs) of each MilliSort phase**

| Phase | 120 servers (0.84M records) | 240 servers (1.68M records) |
|---|---|---|
| Local Sort | 147.0 | 137.8 |
| Partitioning | 200.5 | 410.4 |
| Shuffle | 377.2 | 738.9 |
| Rearrangement | 128.1 | 146.9 |
| Total | 942.3 | 1523.8 |

7000 records/server

2x

2x

remain almost the same

- **Coordination and shuffle costs prevent us from using more servers**
  - Both costs increase with the cluster size (due to small-message throughput)

# Efficiency of MilliSort

- **MilliSort (10 ms) vs. other distributed sorting systems**

| | CPU Model | # HW Threads/core | NetBW/core (Gbps) | Throughput (recs/ms/core) |
|---|---|---|---|---|
| **MilliSort** | Xeon@2.4GHz | 1 | 3.1 | 1297 |
| **TencentSort** | POWER8@2.9GHz | 8 | 5.0 | 1977 |
| **CloudRAMSort** | Xeon@2.9GHz | 2 | 2.7 | 707 |

per-core throughput

**Flash bursts are efficient despite running at millisecond timescales.**

# Discussion

- **Is 1-10 ms the right target?**
  - >100 ms just to communicate with the datacenter over WAN today
  - New edge computing offerings enable <10 ms latency
- **Potential applications?**
  - Real-time decision making without humans in the loop
  - e.g., controllers for IoT devices, financial applications, etc.
- **Limitations/future work**
  - Low duty cycles: colocate flash bursts with batch jobs to achieve high CPU utilization
  - Tackle the problem of loading application data
  - General-purpose infrastructure for executing flash bursts (storage systems, cluster schedulers, networking infrastructure, etc.)

# Conclusion

- **Flash burst is feasible for several core patterns in data analytics**
  - MilliSort and MilliQuery can harness >100 servers in 1 ms
  - Quite efficient despite running in milliseconds
- **Small message throughput is the primary limiting factor to scalability**
  - At least equally important as latency and network bandwidth in flash bursts
- **We hope our results will spark interests in flash bursts**
  - Encourage application developers to explore practical usage of flash bursts

# Questions

Contacts:
yilongl@cs.stanford.edu
seojin@csail.mit.edu