ADOC: Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance

Jinghuan Yu[1]

Sam H.Noh[2,3], Young-ri Choi[2], Chun Jason Xue[1]

[1] City University of Hong Kong

[2] UNIST(Ulsan National Institute of Science & Technology)

[3] Virginia Tech







Research Background

- Working scenarios of LSM-KV
- Common architecture of LSM-KV
- Write stall: a common problem

Review former conclusions

Data Overflow: root cause of write stalls

ADOC: avoid write stalls by controlling data flow

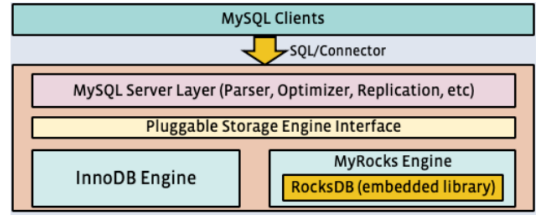
Working scenarios of LSM-KV

ZippyDB

KV store clusters

Log-Structured Merge-Tree:

- Write-optimized
- Out-of-place updates
- Sequential batch operations



DB engine

LogDevice

Log storage server



RocksDB[1] is the most popular LSM-KV with wide usage in different scenarios

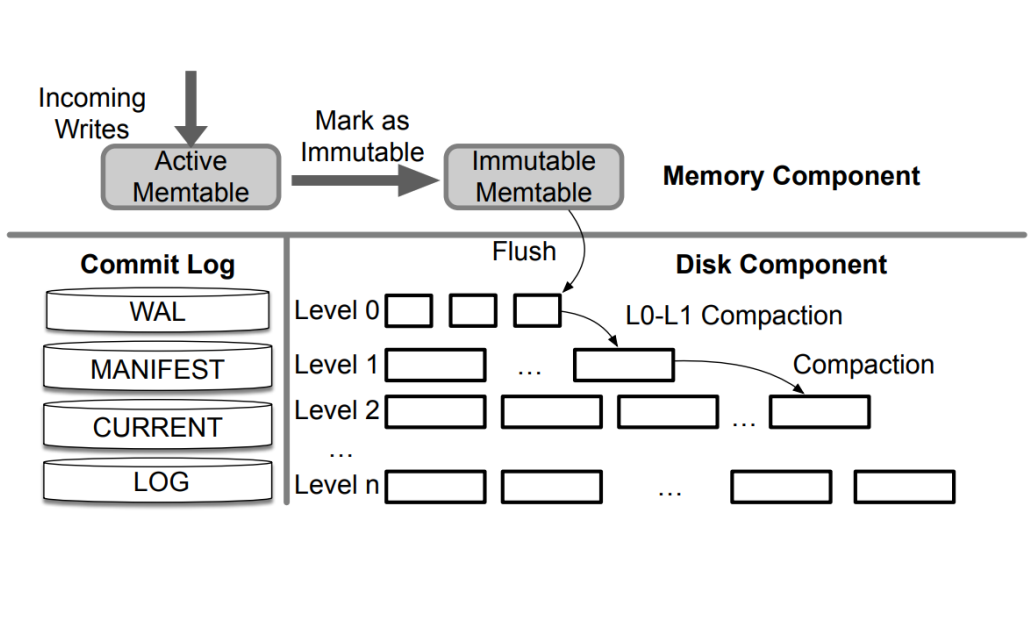
diem

Blockchain

[1] https://github.com/facebook/rocksdb/wiki

Common architecture of LSM-KV

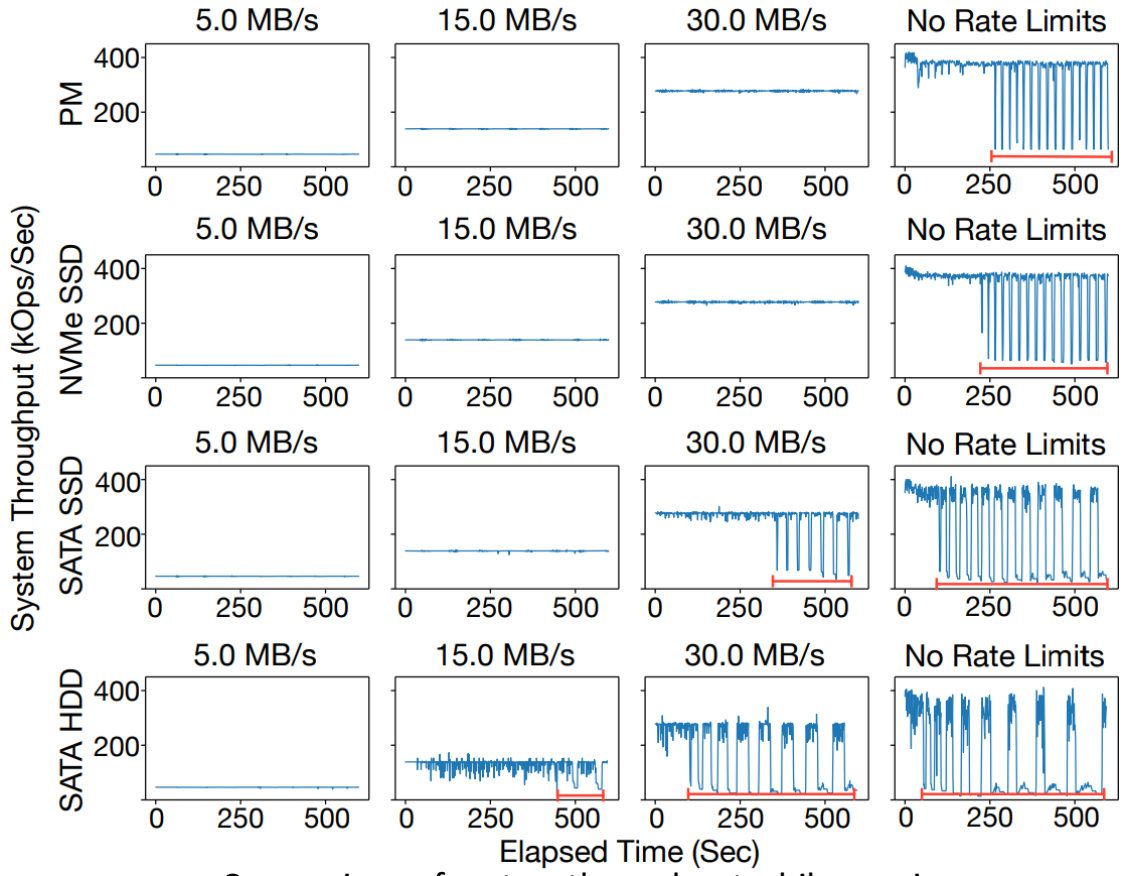
- **Common architecture:**
  - Memory component
    - Active Memtable
    - Immutable Memtable
  - Commit Log
  - Disk component
    - Multiple level of SSTs



- **Major operations:**
  - Flush: Transform Memtable into SSTs in Level 0 (L0)
    - L0 is different from other levels ➔unsorted and overlapped
  - Compaction: Merge SSTs from shallower levels to deeper levels
    - L0-L1 compaction
    - Deep-level compaction

Write stall: a common problem

- **Write stall:**
  - Sudden drop in system performance
  - Mentioned by many prior studies
- **Observed characteristics:**
  - **Universal:**
    - Write stalls happen on all devices
  - **Device dependent:**
    - Duration, frequency of occurrence and rate of performance degradation vary between devices



Comparison of system throughput while running *fillrandom* workload with increasing write pressure

Experiment setups in this paper:
- 2 Intel Xeon® Gold 6230
- 128 GB DDR4 DRAM
- *fillrandom* workload

Device	Product Name	Device Capacity	Sequential Bandwidth
PM	Optane DC PMM	512 GB	2300 MB/s
NVMe SSD	Samsung 970 PRO	1 TB	2700 MB/s
SATA SSD	Intel DC S4500	960 GB	490 MB/s
SATA HDD	Seagate ST1000DM010	1 TB	210 MB/s

Goal of this study

- **Understand underlying cause of write stalls**
  - Along ALL types of storage devices
- **Find solution to remove/alleviate write stalls**
  - For ALL types of storage devices

Research Background

Review existing conclusions

- Prior studies on write stalls
- Inconsistent points of former conclusions

Data Overflow: root cause of write stalls

ADOC: avoid write stalls by controlling data flow

Prior studies on write stalls

Resource exhaustion[1-10]:

- Why are write stalls happening when there are idle resources?

Execution of L0-L1 compaction[6,7]:

- Why are occurrences of L0-L1 compaction and compaction jobs not aligned?

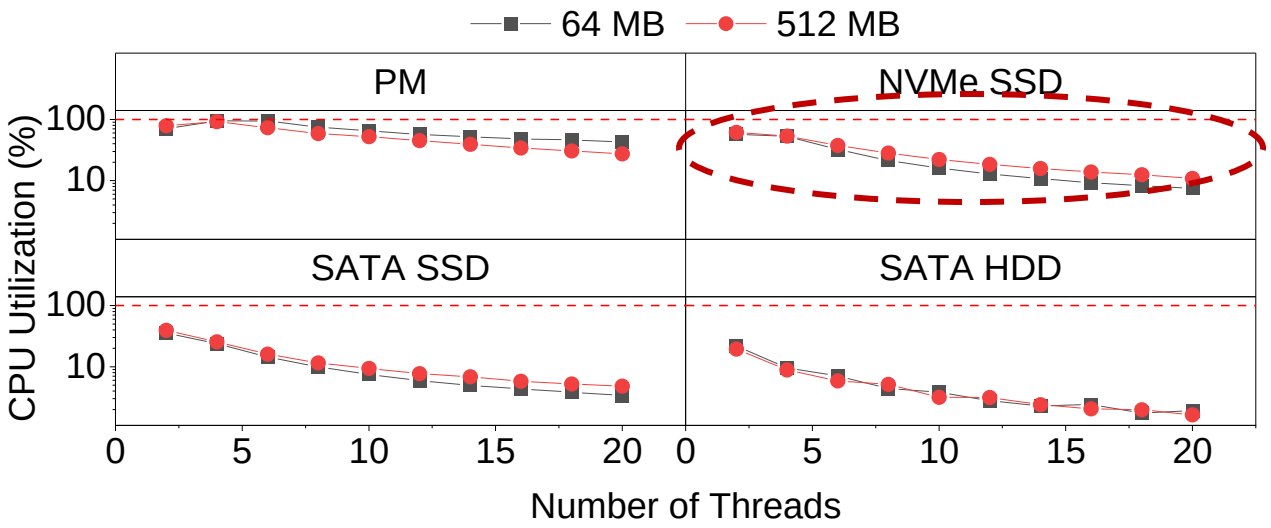
Crowded compaction jobs[8,9,10]:

- Why do write stall durations and occurrences decrease when there are more threads?

[1] From Flash to 3D XPoint (ISPASS 2020)
[2] KVell (SOSP 2019)
[3] FPGA-based Compaction Engine for Accelerating LSM-tree Key-Value Stores (FAST 2020)
[4] FPGA-Accelerated Compactions for LSM-based KeyValue Store (ICDE 2020)
[5] The RocksDB Experience (FAST 2021)
[6] SILK (ATC 2019)
[7] MatrixKV (ATC 2020)
[8] On performance stability in LSM-based storage system (VLDB 2019)
[9] PebblesDB (SOSP 2017)
[10] bLSM (Sigmod 2012)

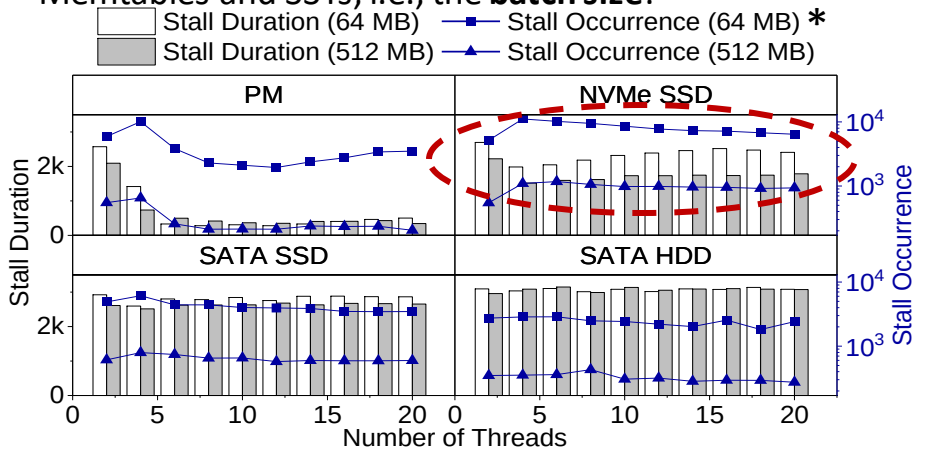
Resource Exhaustion:

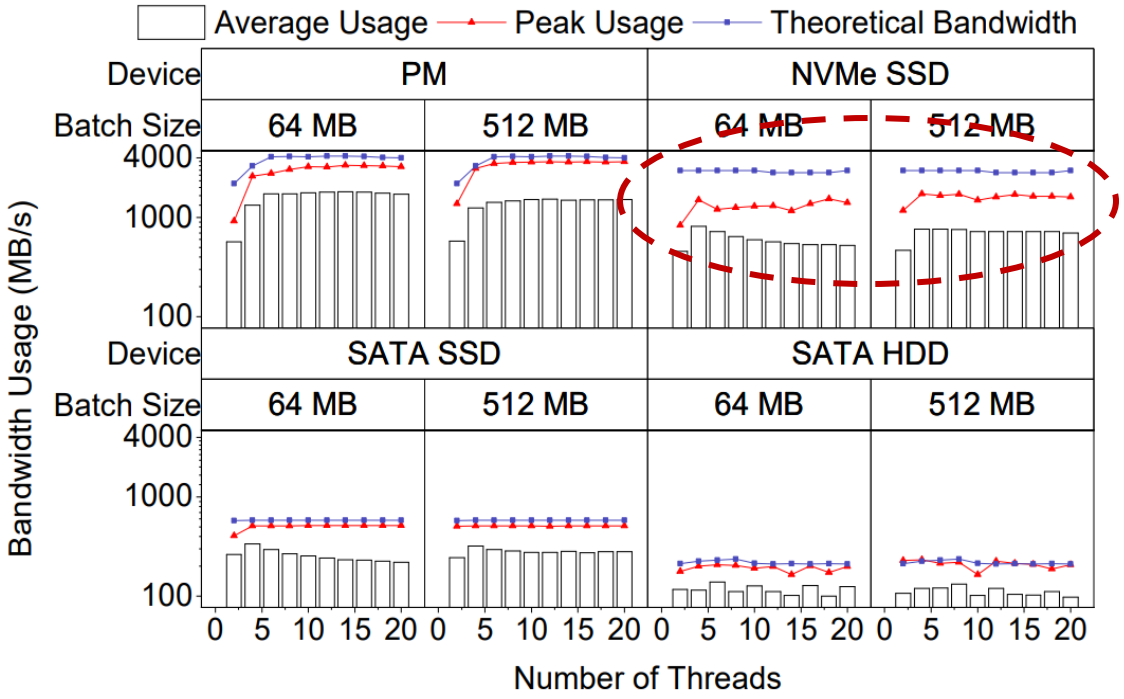
Why are write stalls happening when there are idle resources?



- **Normalized CPU utilization decreases with increased number of threads.**

* 64 MB and 512 MB refer to the common size of Memtables and SSTs, i.e., the **batch size**.

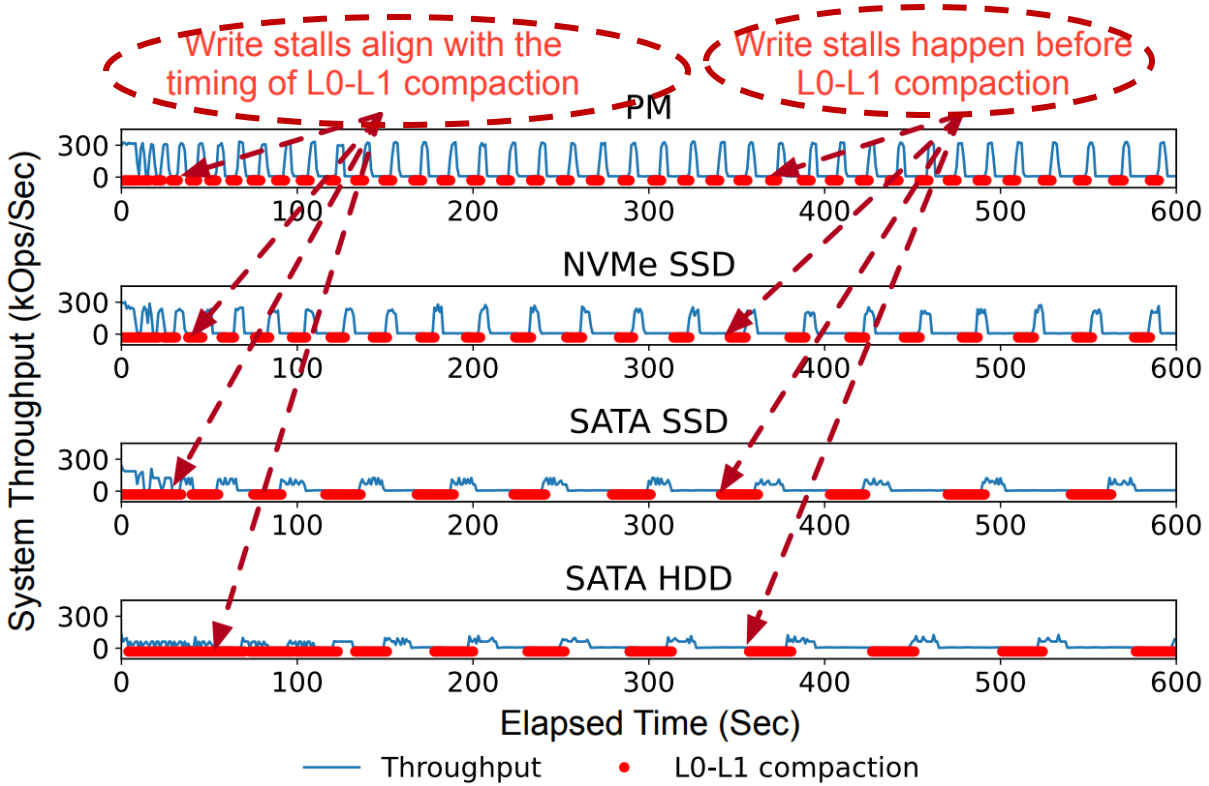




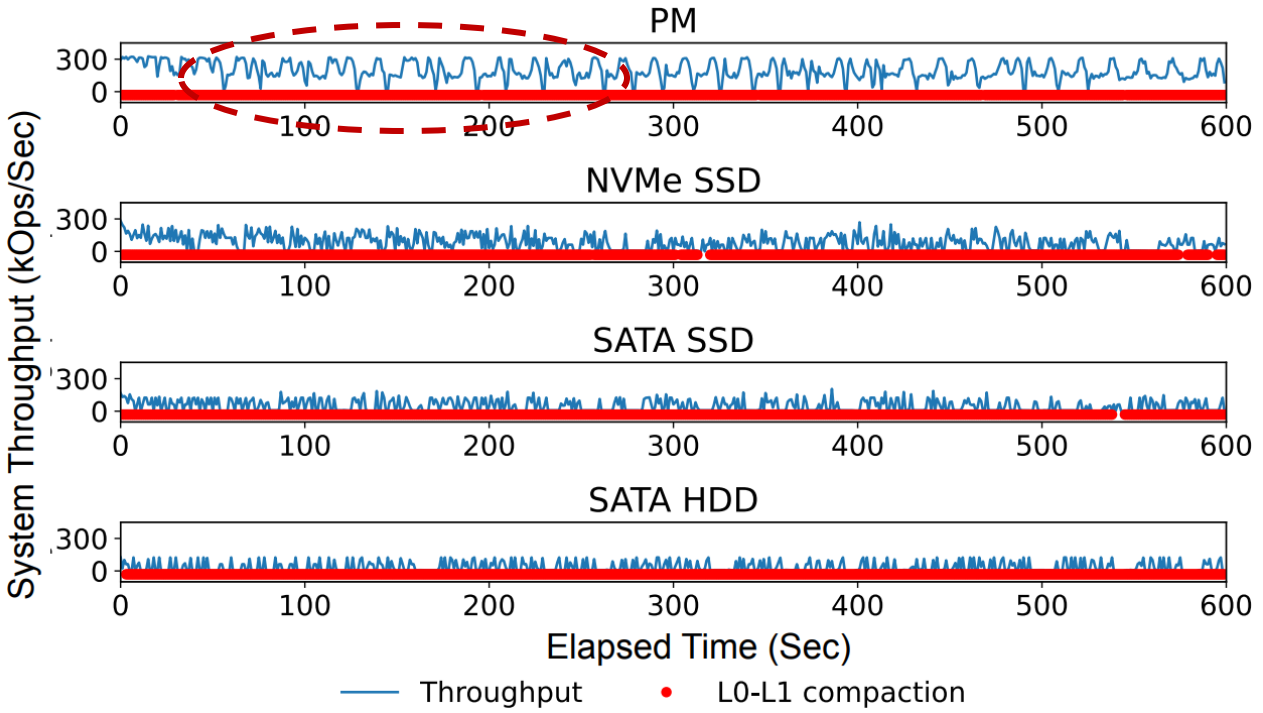
- **Bandwidth usage does not reach theoretical bandwidth.**

Execution of L0-L1 Compactions:

Why are occurrences of L0-L1 compaction and compaction jobs not aligned?



(a) System throughput with default configuration

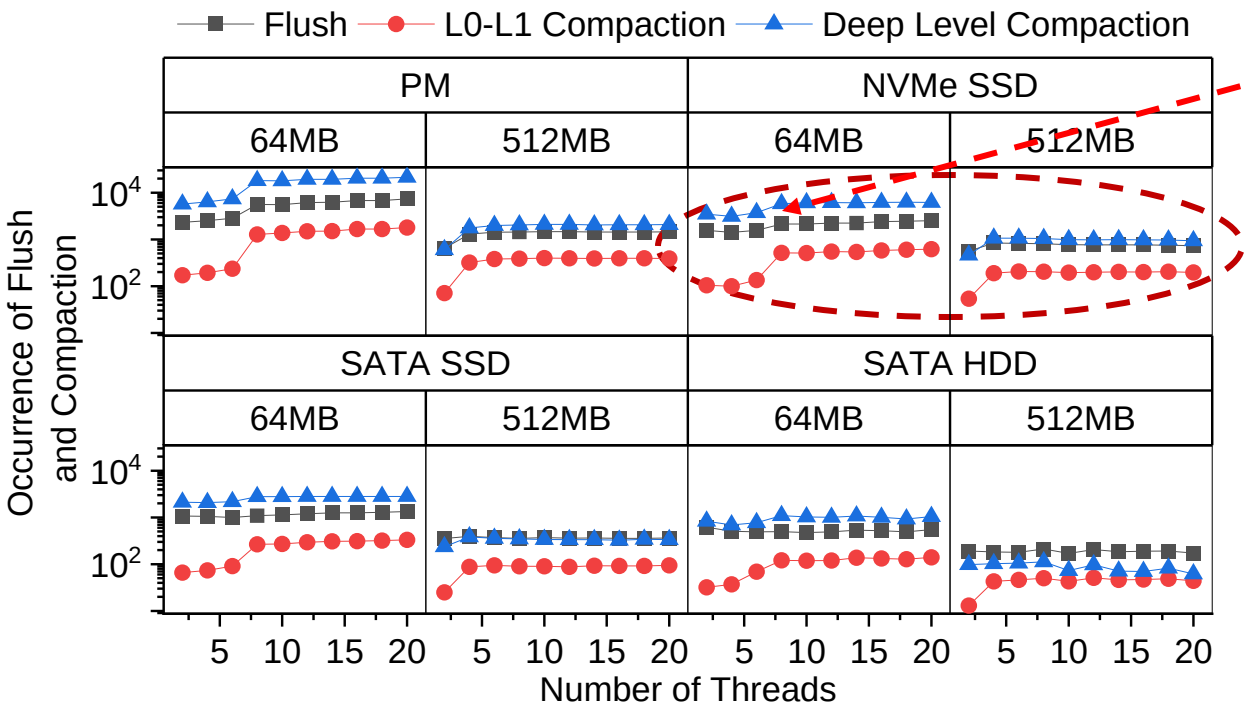


(b) System throughput with 20 threads

- **Execution of L0-L1 compaction and performance drop hardly match**

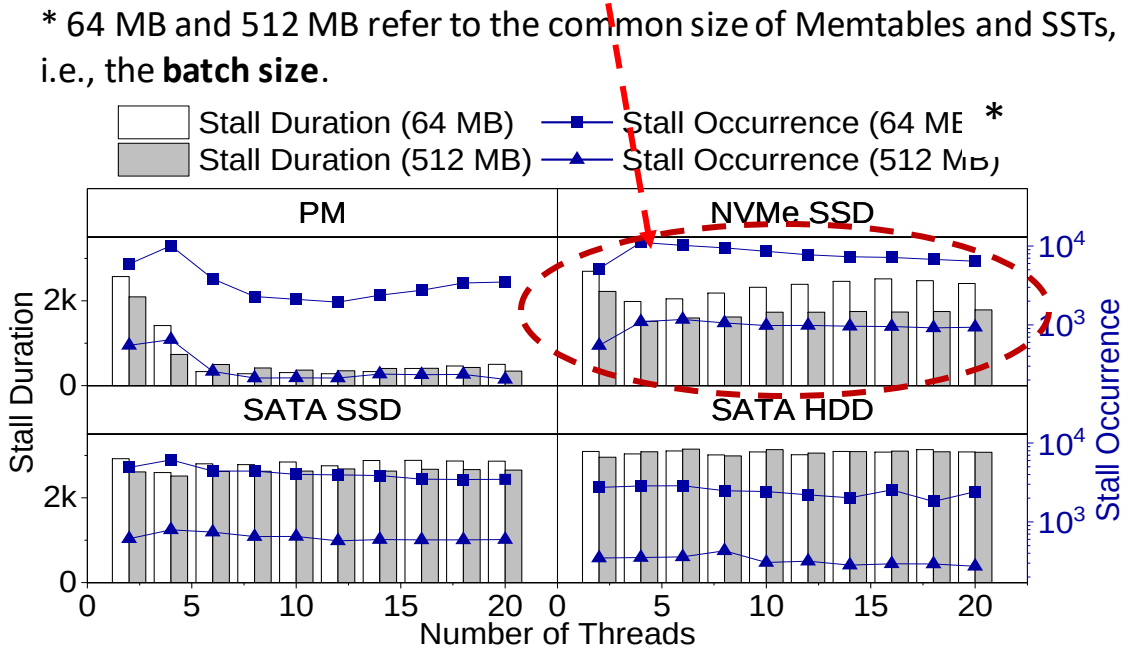
Crowded Compaction Jobs:

Why do write stall durations and occurrences decrease when there are more threads?



(a) Occurrences of background jobs

- Keep increasing till the number of threads reaches 8.
- Stall occurrence decreases from 6



(b) Stall occurrences with 512 MB batches

- **The number of background jobs stops increasing**
- **The duration and occurrence decreases**

Research Background

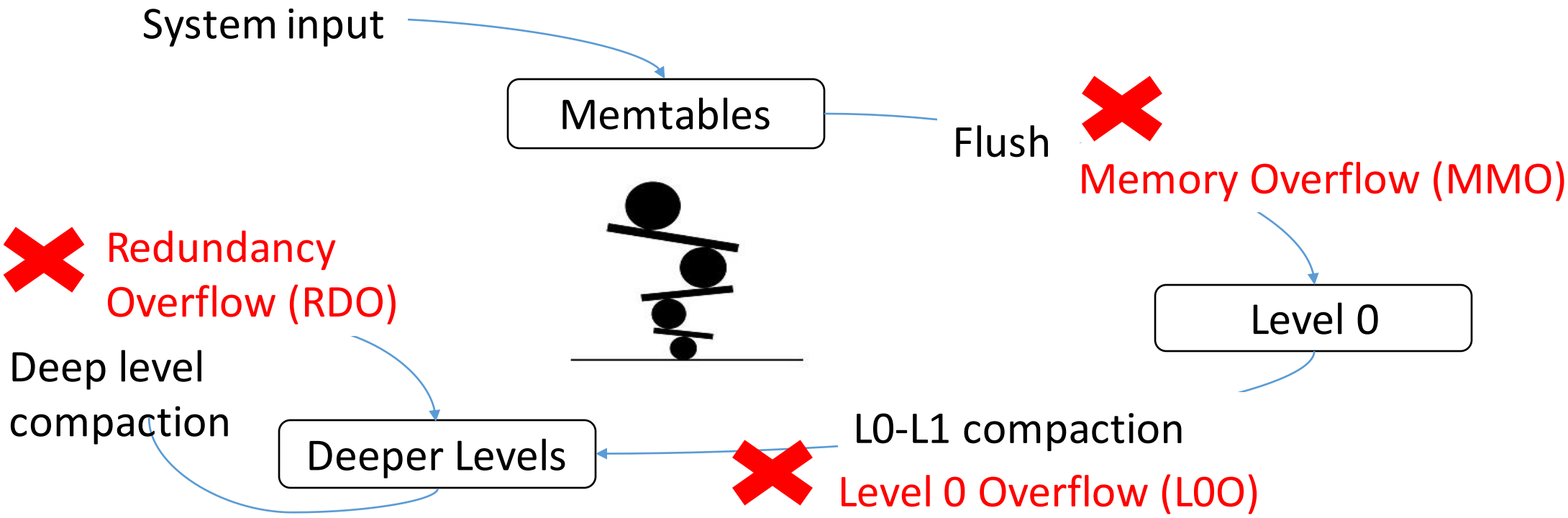
Review former conclusions

Data Overflow: root cause of write stalls

- Definition and consequences of data overflow
- Explaining the unexplained

ADOC: avoid write stalls by controlling data flow

Harmony between dataflows



- Balance breaks, harmony collapses, overflow occurs, and causes write stall
- More than disk overflow:
  - Not only limited by the disk, but also related to the traversal, schedule policy and parallelism, etc.

Data overflow: root cause of write stalls

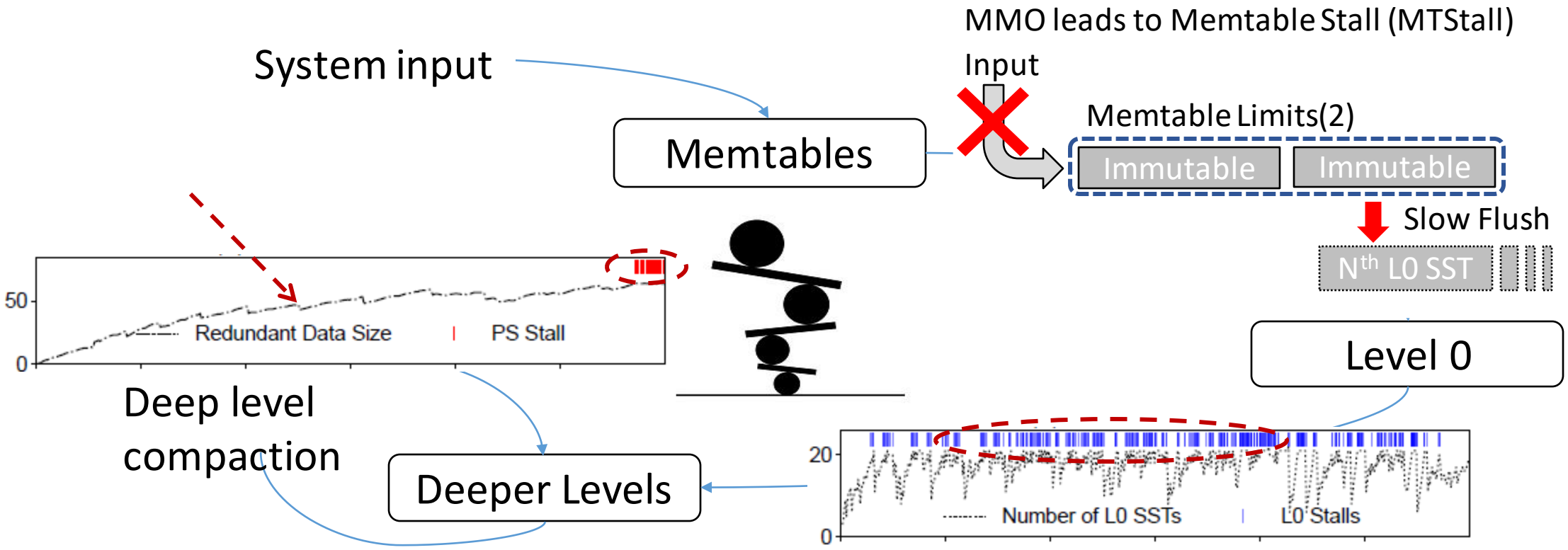
- **Data overflow:**
  - Input rate of one or more components is higher than the output rate
    - which is determined by the processing rate of related data movement jobs
- **Consequence of data overflow:**
  - Related components will expand rapidly and exceed system limitation

➔ system stops input to avoid further data accumulation within component

➔ causes write stalls

Data overflow	Related component(s)	Input stream	Related data movement jobs	Result
Memory Overflow (MMO)	Memtables	System input	Flush job	MT Stall
Level 0 Overflow (L0O)	Level 0 SSTs	Flush job	L0-L1 compaction job	L0 Stall
Redundancy Overflow (RDO)	Deep level SSTs	Compaction jobs in shallower levels	Compaction jobs in deeper levels	PS Stall

How data overflow leads to write stalls



Data overflow	Related component(s)	Result
Memory Overflow (MMO)	Memtables	Memtable Stall (MT Stall)
Level 0 Overflow (L0O)	Level 0 SSTs	Level 0 Stall (L0 Stall)
Redundancy Overflow (RDO)	Deep Level SSTs	Pending Compaction Input Size (PS Stall)

Explaining the unexplained

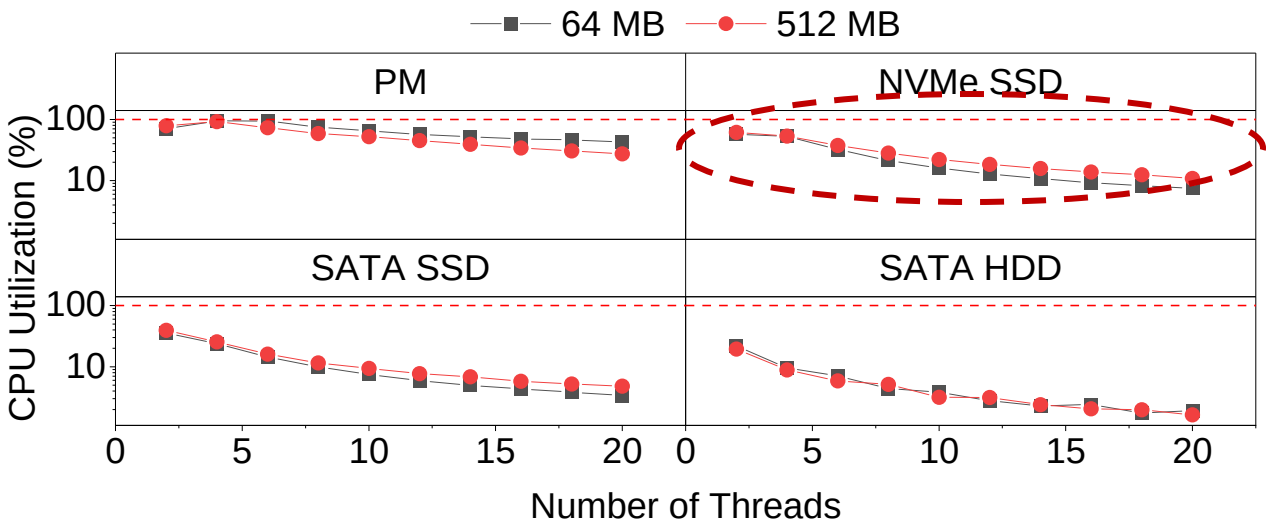
Resource Exhaustion:
Why are write stalls happening when there are idle resources?

Execution of L0-L1 Compactions:
Why are occurrences of L0-L1 compaction and compaction jobs not aligned?

Crowded Compaction Jobs:
Why do write stall durations and occurrences decrease when there are more threads?

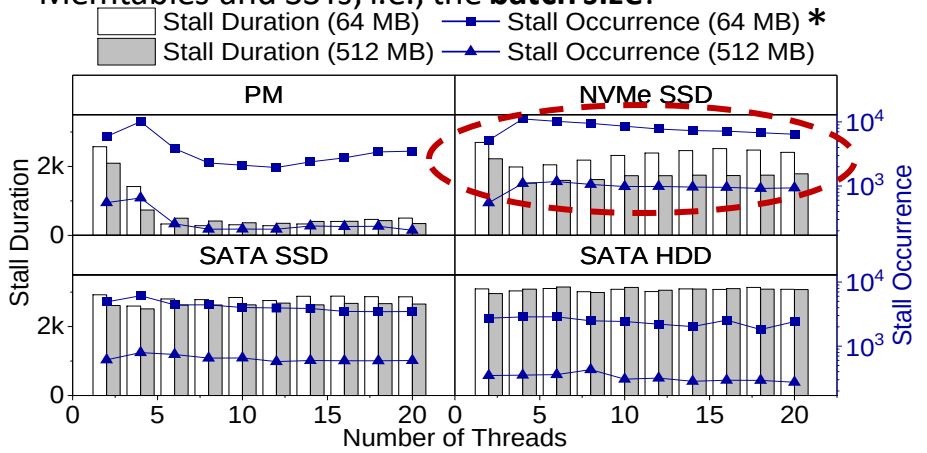
Resource Exhaustion:

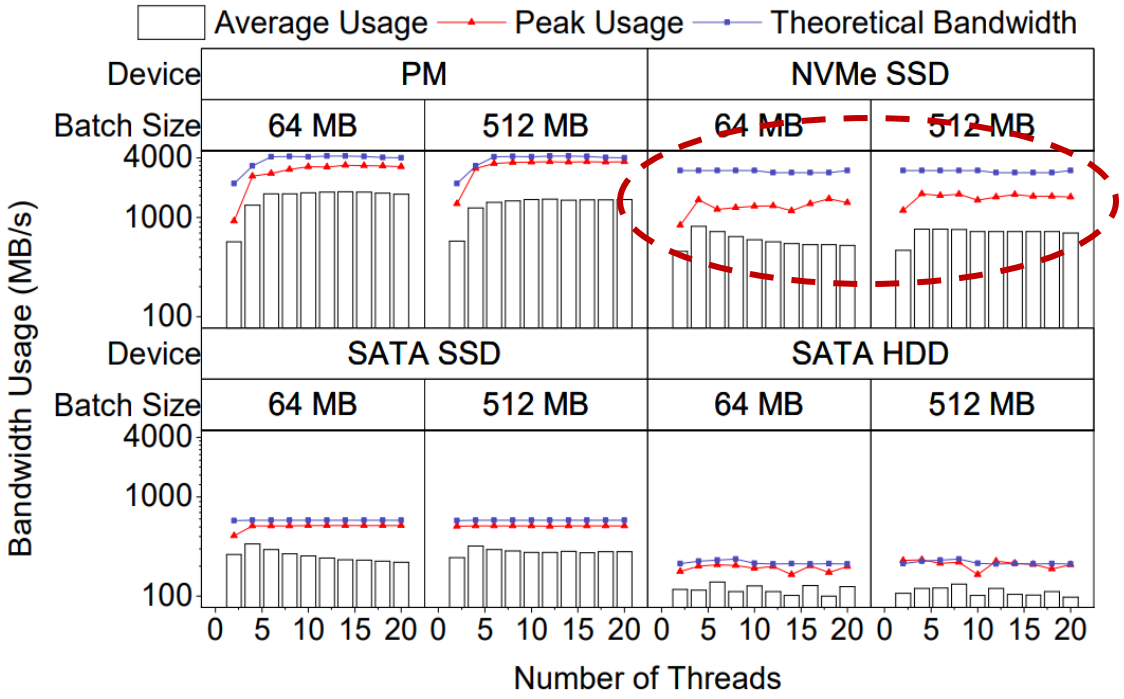
Why are write stalls happening when there are idle resources?



- **Normalized CPU utilization decreases with increased number of threads.**

* 64 MB and 512 MB refer to the common size of Memtables and SSTs, i.e., the **batch size**.

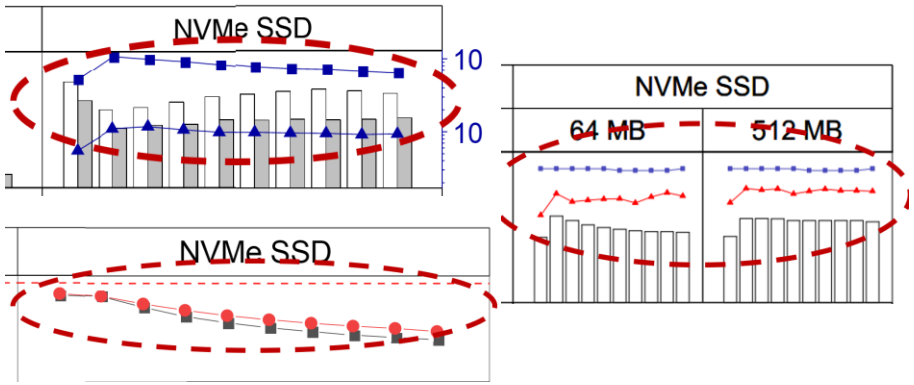


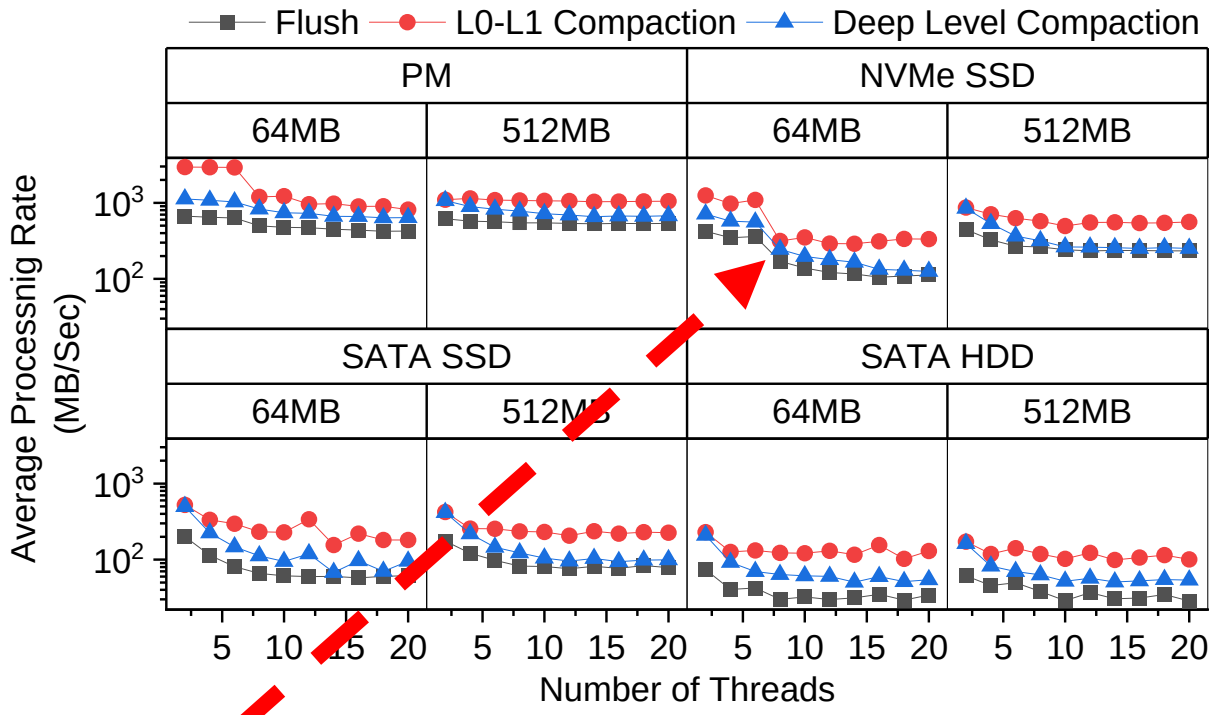


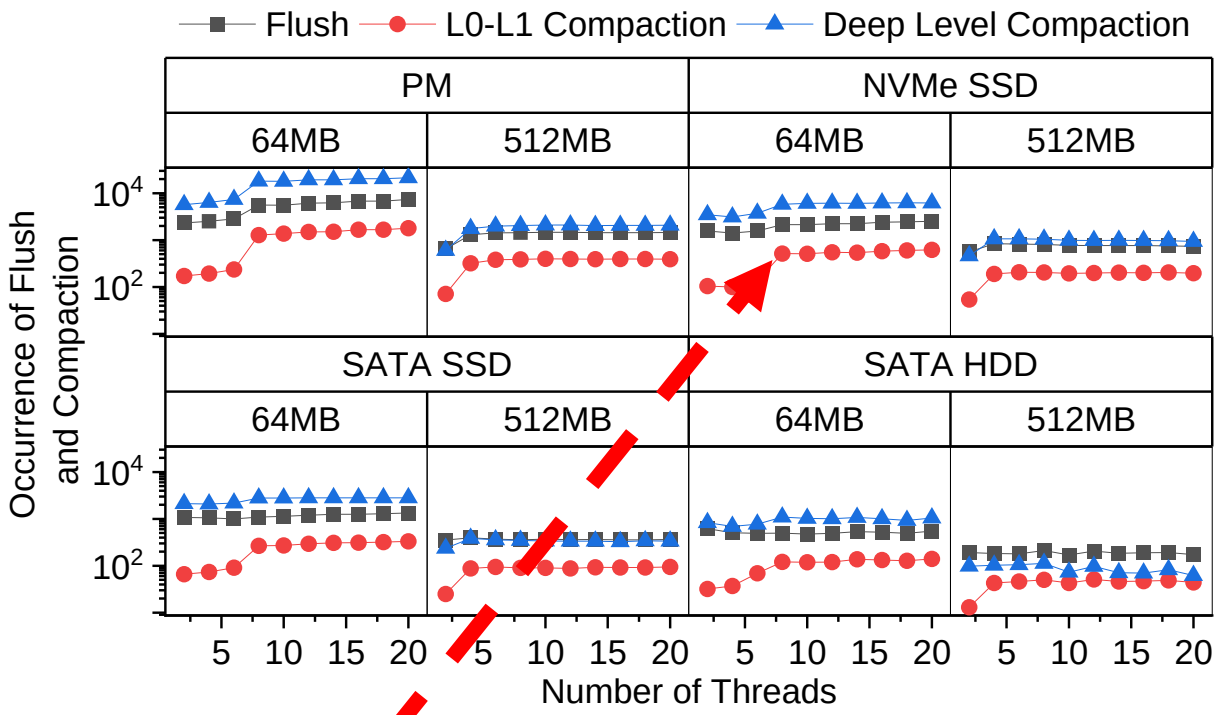
- **Bandwidth usage does not reach theoretical bandwidth.**

Resource Exhaustion:

Why are write stalls happening when there are idle resources?





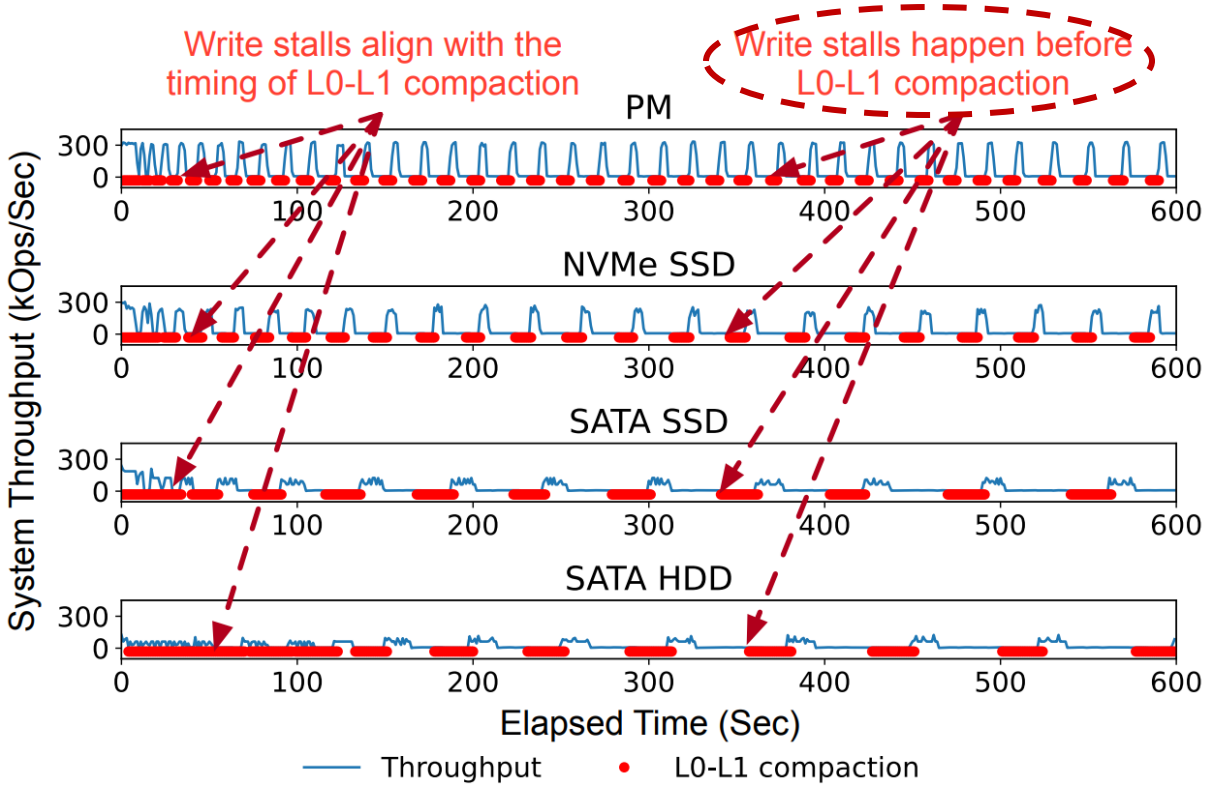


- Increasing threads will lead to slower flush rate, forms **MMO**
- More MT Stalls occurs in the system, limit the total input data size
- The scheduling/execution frequency of background jobs reaches limit

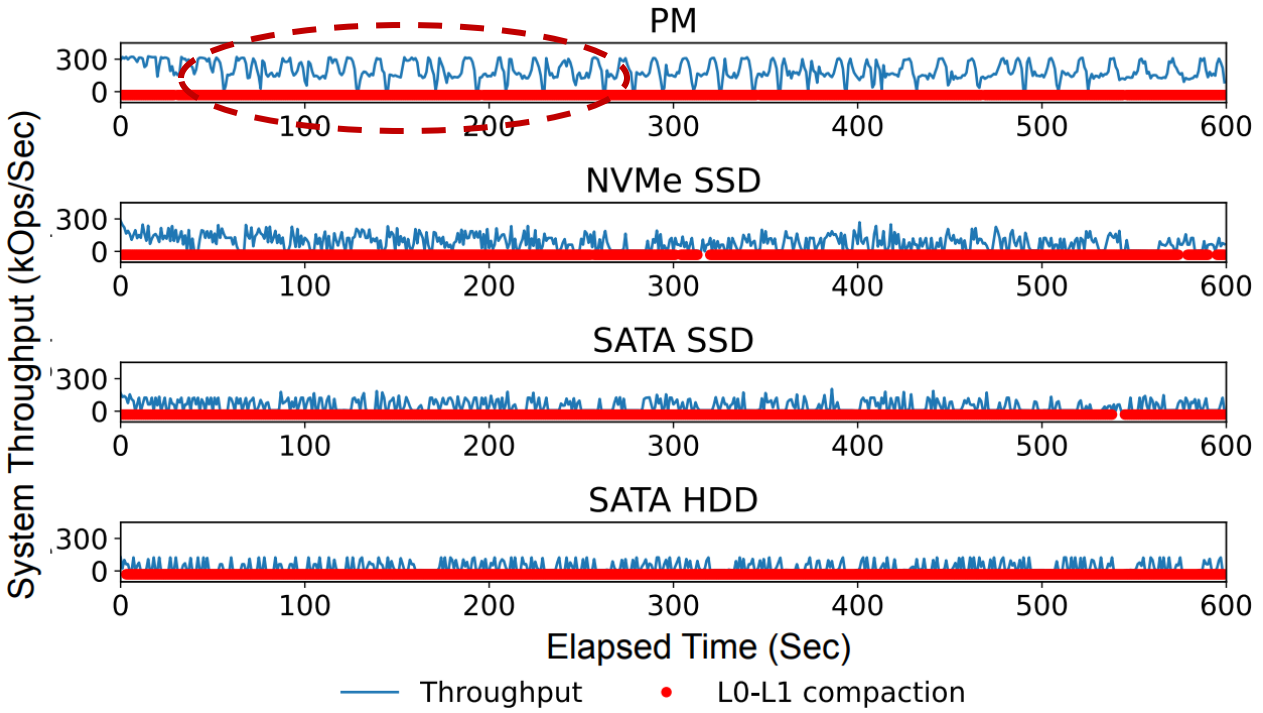
➜ CPU and disk remain idle

Execution of L0-L1 Compactions:

Why are occurrences of L0-L1 compaction and compaction jobs not aligned?



(a) System throughput with default configuration

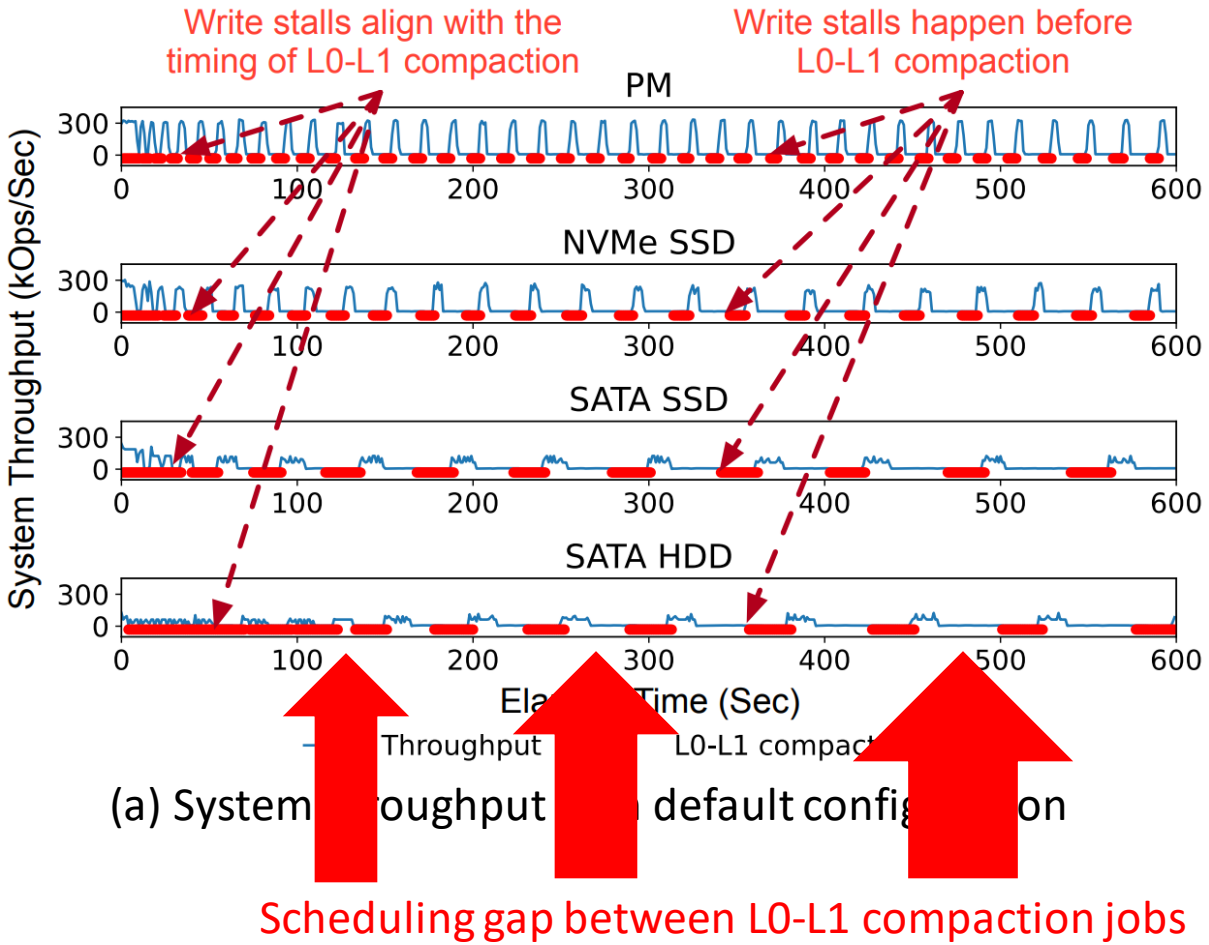


(b) System throughput with 20 threads

- **Execution of L0-L1 compaction and performance drop hardly match**

Execution of L0-L1 Compactions:

Why are occurrences of L0-L1 compaction and compaction jobs not aligned?



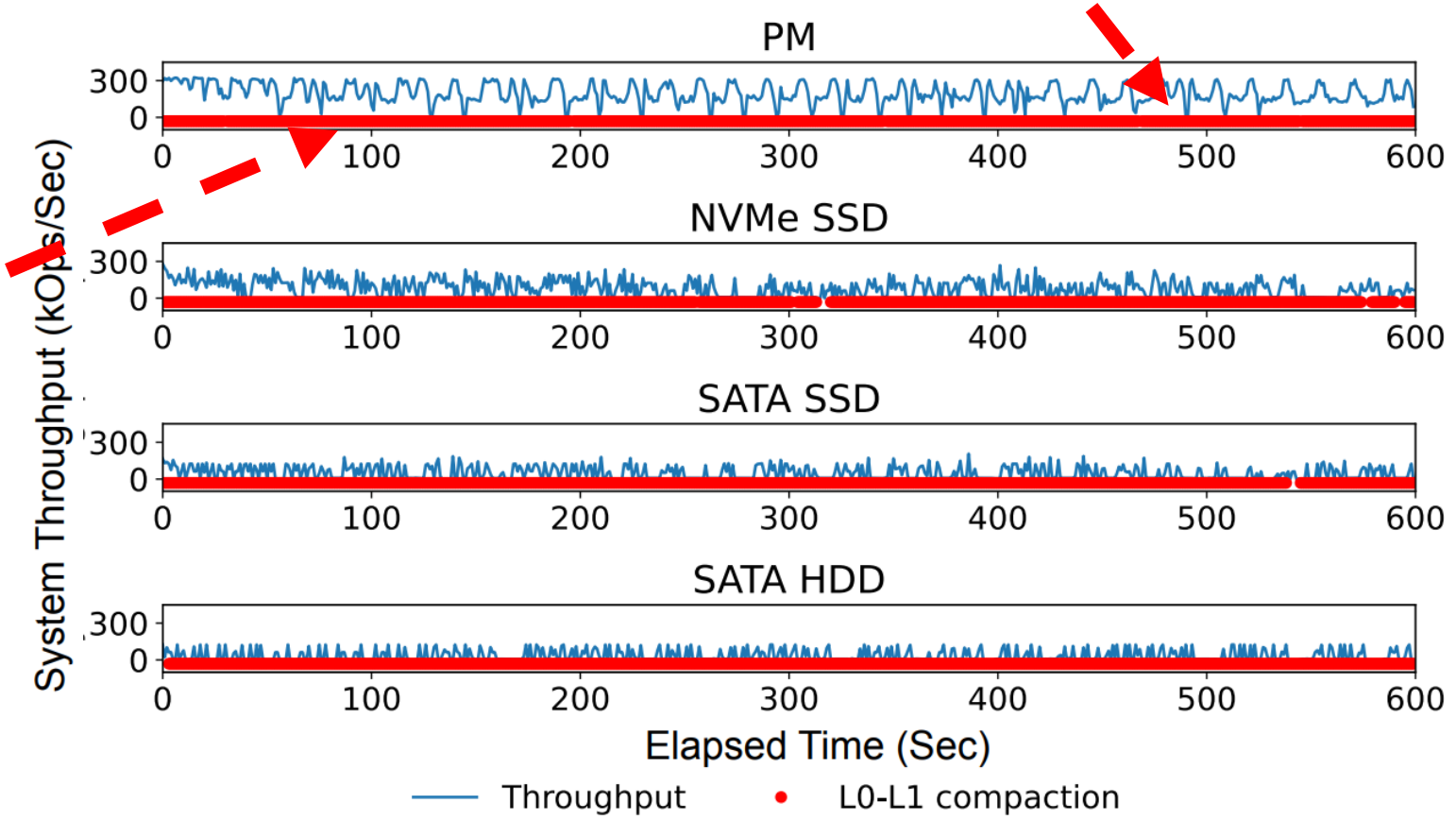
- Only one thread for all compactions
  - ➜ L0-L1 compaction can not get executed in time
  - ➜ L0 SSTs are accumulating and **L0O** occurs
- Compaction duration increases after running
  - ➜ Scheduling gap between L0-L1 compaction jobs increases
  - ➜ L0 Stalls before L0-L1 compaction jobs get executed.

Execution of L0-L1 Compactions:

Why are occurrences of L0-L1 compaction and compaction jobs not aligned?

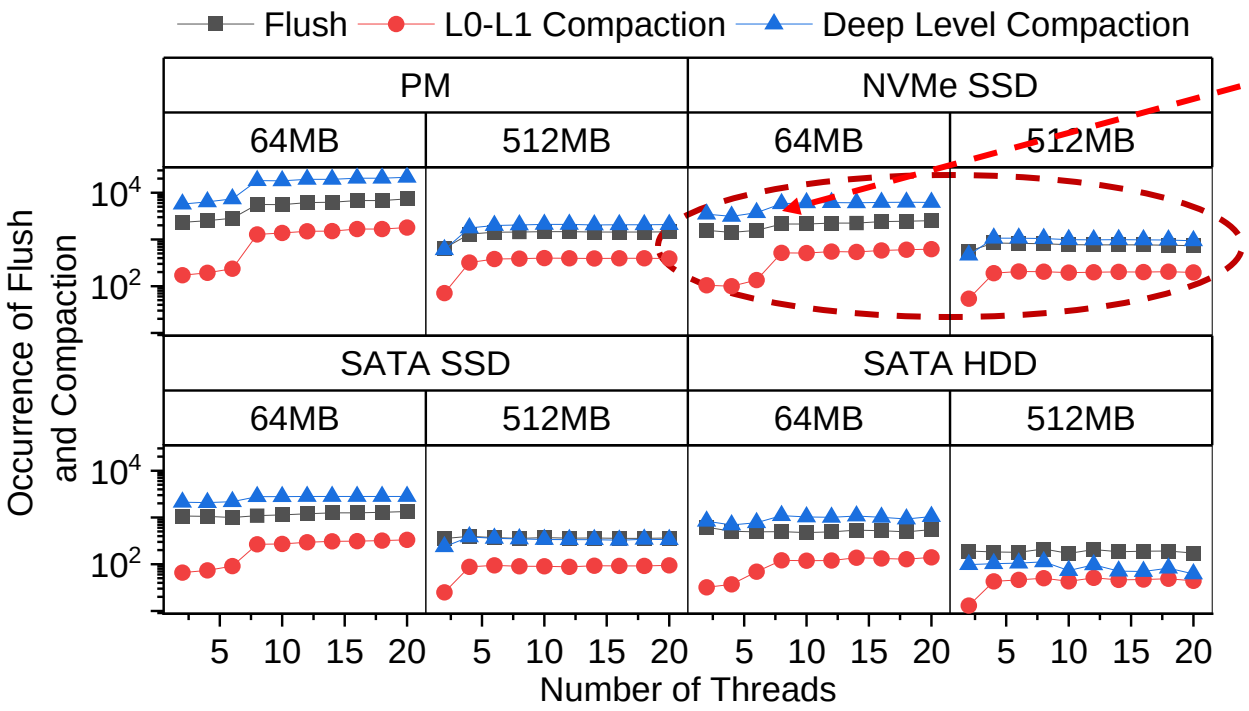
- Enough number of threads
- L0-L1 compaction jobs get executed immediately
  - ➜ L0-L1 compaction occurs continuously
- Flush rate gets decreased
  - ➜ MMO occurs and blocks the system
  - ➜ More write stalls with shorter duration

Much more regular, with shorter duration



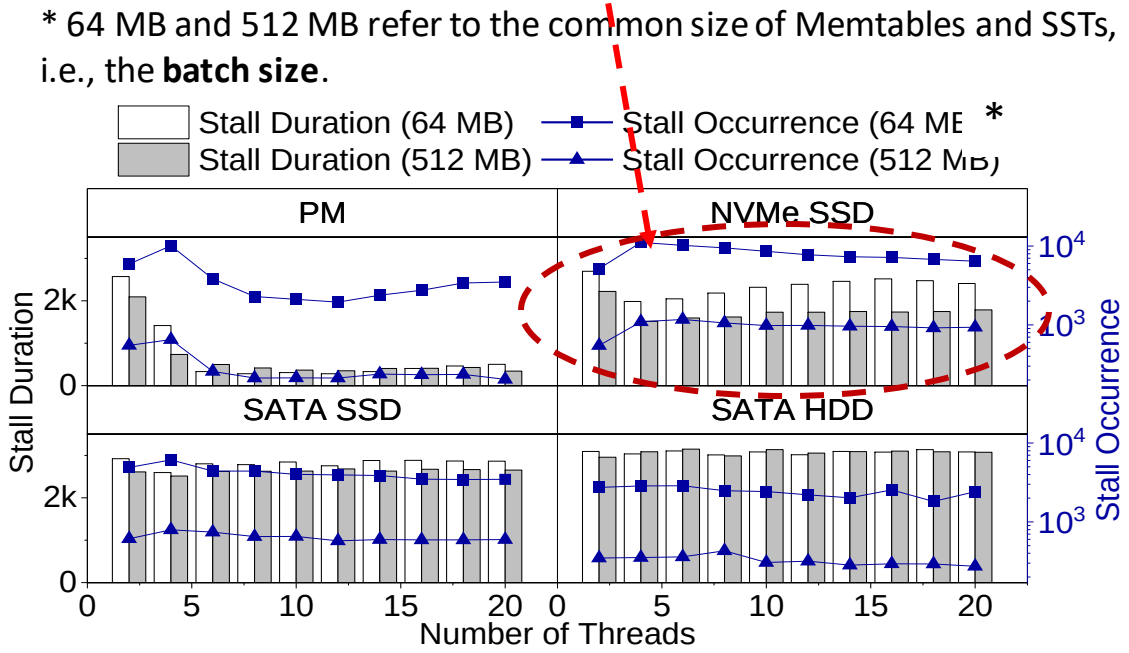
Crowded Compaction Jobs:

Why do write stall durations and occurrences decrease when there are more threads?



(a) Occurrences of background jobs

- Keep increasing till the number of threads reaches 8.
- Stall occurrence decreases from 6

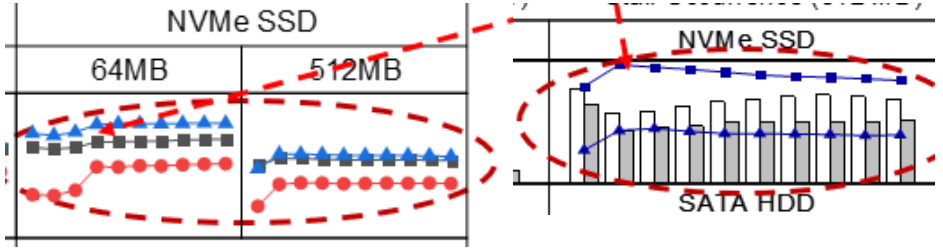


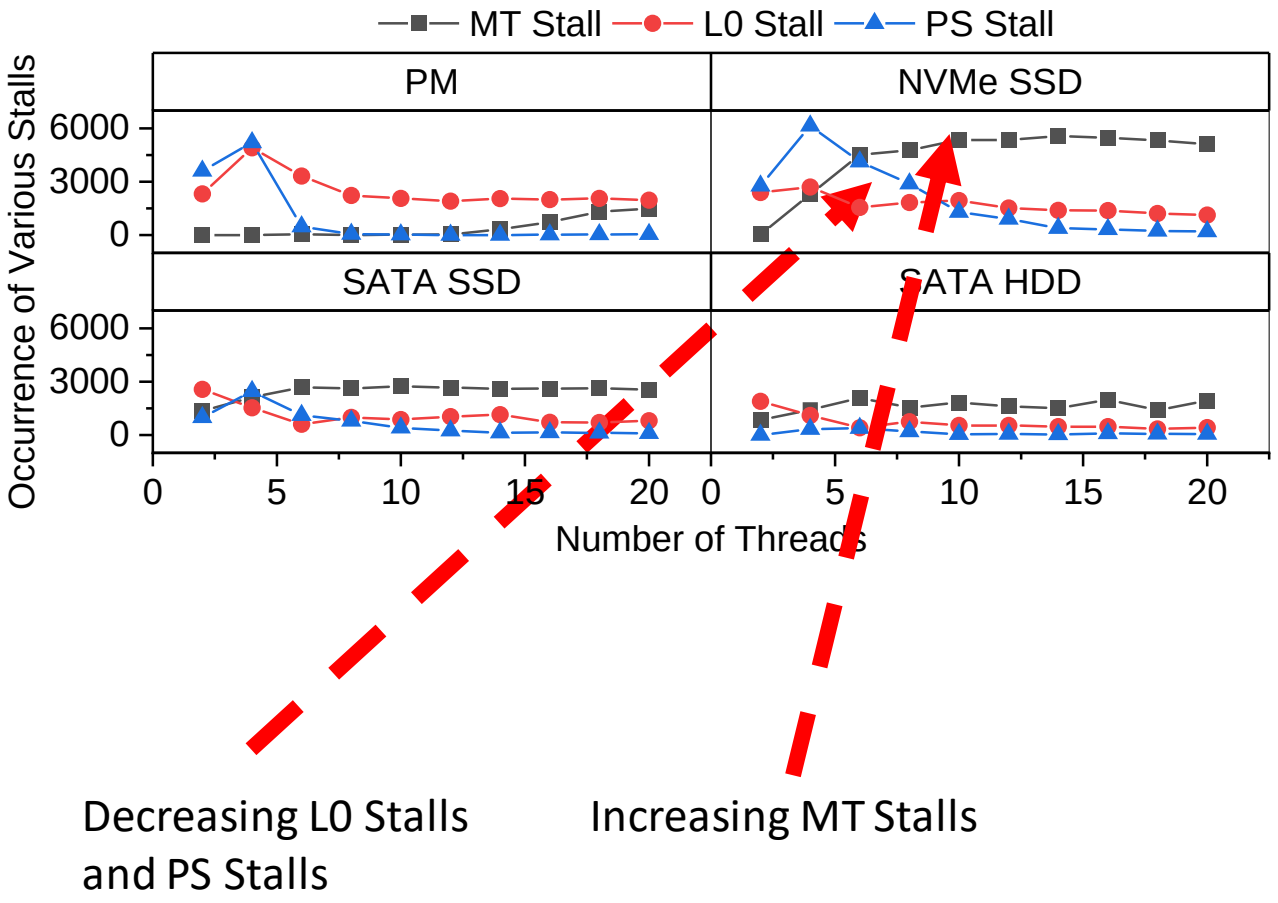
(b) Stall occurrences with 512 MB batches

- **The number of background jobs stops increasing**
- **The duration and occurrence decreases**

Crowded Compaction Jobs:

Why do write stall durations and occurrences decrease when there are more threads?





Stall duration is determined by:

- Large scale, longer duration:
  - L0-L1 >> Deep compaction > Flush
- Worse parallelism, longer duration:
  - Flush < L0-L1 << Deep compaction

➜ **Stall duration order**

- L0O>RDO>MMO
- Slower flush rate
  - ➜ More MMO, less L0O
- Higher parallelism
  - ➜ less RDO

➜ Reduction in total stall duration

Goal of this study

- **Understand underlying cause of write stalls**
  - Along ALL types of storage devices

➜ **Data Overflow: mismatch in data flow among LSM KV components**

- **Find solution to remove/alleviate write stalls**
  - For ALL types of storage devices

➜ **ADOC: Automatic Data Overflow Control**

Research Background

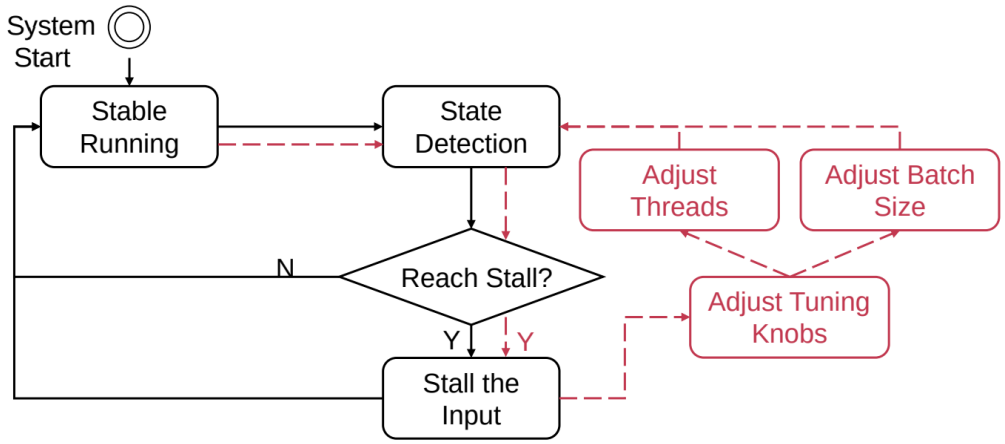
Review former conclusions

Data Overflow: root cause of write stalls

ADOC: avoid write stalls by controlling data flow

- Workflow of ADOC
- Design advantages of ADOC
- Evaluation of ADOC

Workflow of ADOC



- L0O: increase threads
- RDO: increase threads and decrease batch size
- MMO: decrease threads and increase batch size

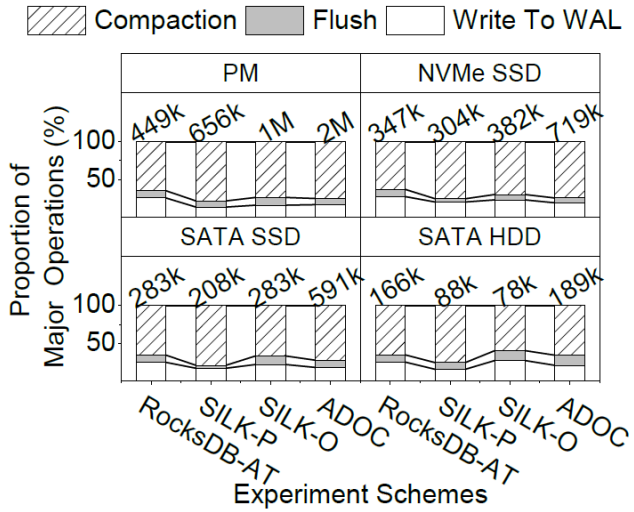
- **Automatic Data Overflow Control (ADOC)**
  - An extended framework of RocksDB
- **Two tuning knobs:**
  - Number of threads
  - Batch size
- **Solving order:**
  - L0O
  - RDO
  - MMO
- **Additive-Increase/Multiplicative-Decrease(AIMD)-based tuning algorithm:**
  - Slow increase to explore suitable configuration
  - Rapid decrease to avoid competition
  - Commonly used in TCP-IP

Design advantages of ADOC

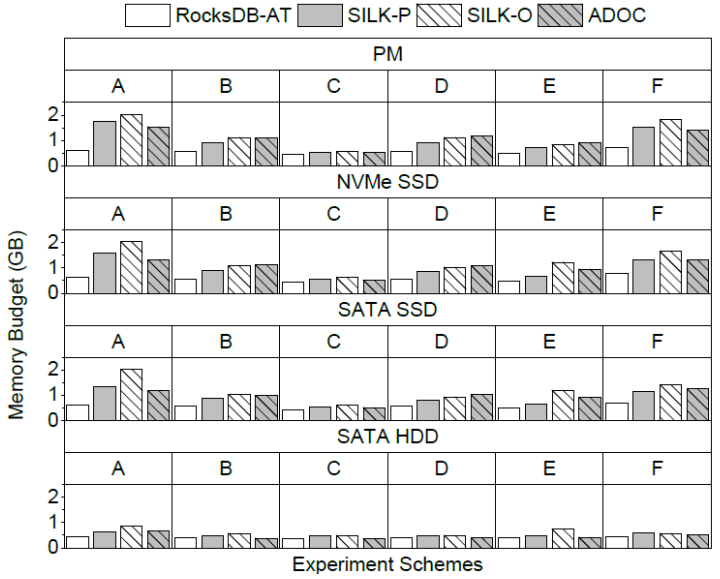
Type of write stall	**Optimization operations**		
	ADOC	SILK[1]	RocksDB-AT[2]
Stall avoidance	+ Optimize on all kinds of write stalls + Will not limit the bandwidth	+ Preemptive scheduling (MT and L0 Stall) - Hard limit on the bandwidth	+ Avoid background IO spikes (MT Stall only) - Hard limit on the bandwidth
Automation level	+ Fully automatic	- Need to provide the bandwidth information - Need manual optimization to find a suitable startup configuration	
Implementation & deployment	+ No intrusive code + Can be deployed with other tuners + Thread safe	- Deep modification (scheduler and rate-limiter) - Bind with RocksDB - Risk of deadlock	+ Embedded in RocksDB

[1] SILK: Preventing Latency Spikes in Log Structured Merge Key-Value Stores (ATC 2019)
[2] Auto-tuned Rate Limiter | RocksDB

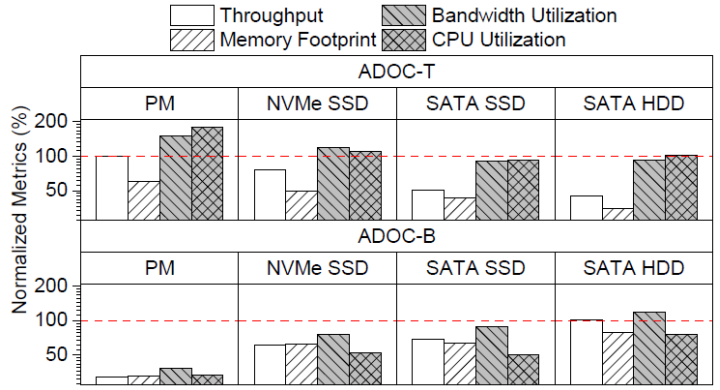
More experiments in the paper:



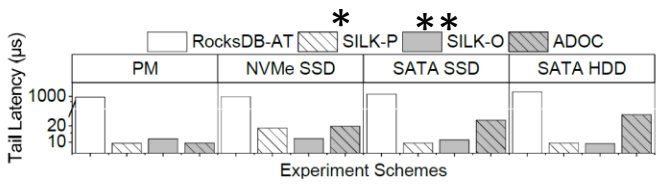
- Comparison of operation proportion



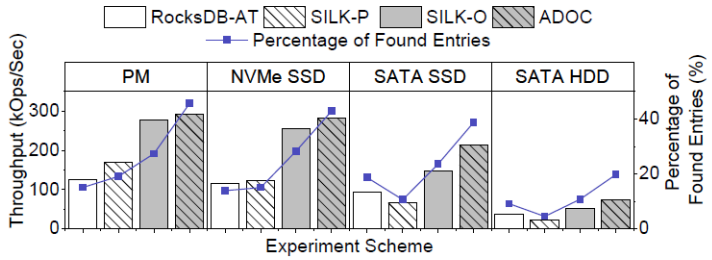
- Comparison of memory usage



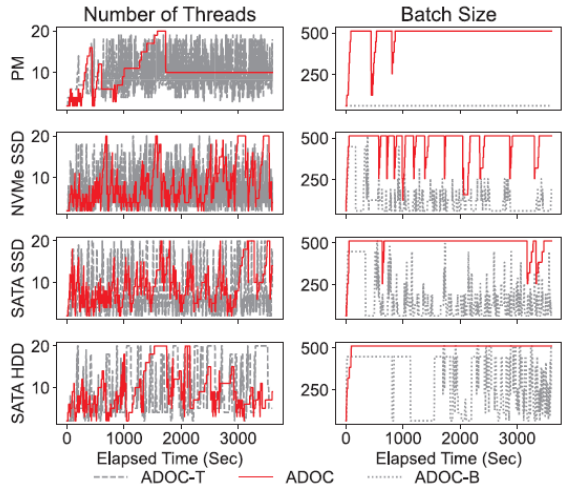
- Comparison of resource usage (up↑) and tuning effect (down↓) of individual components



- Comparison of tail latency



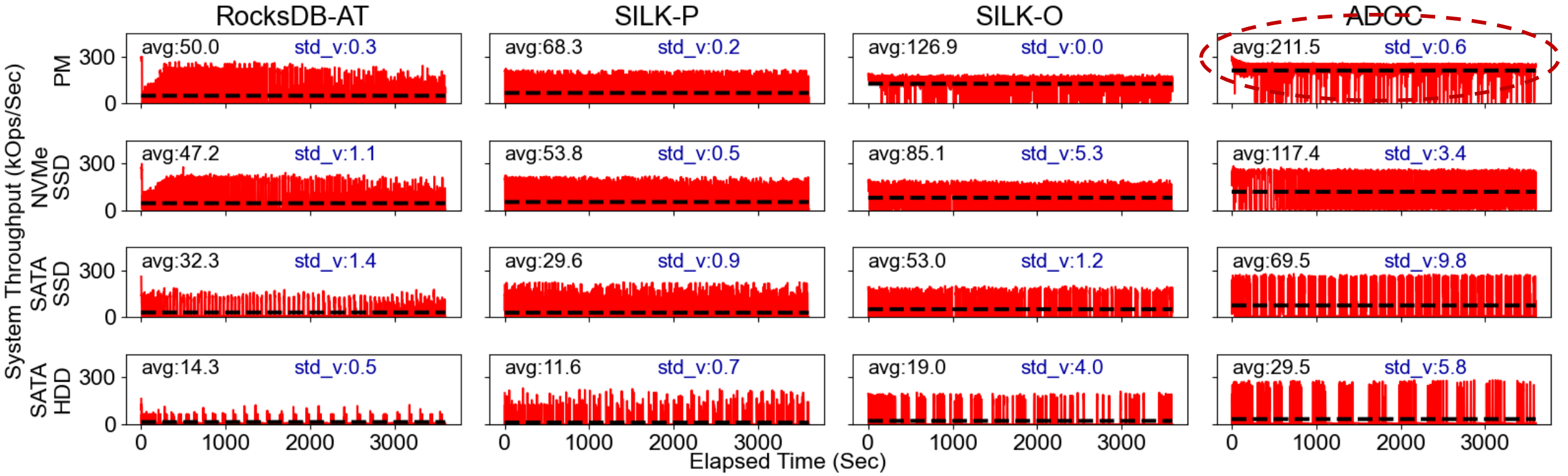
- Comparison of throughput in read and write-intensive workload



***SILK-P (SILK with 4 threads, 128 MB, used in SILK paper)**

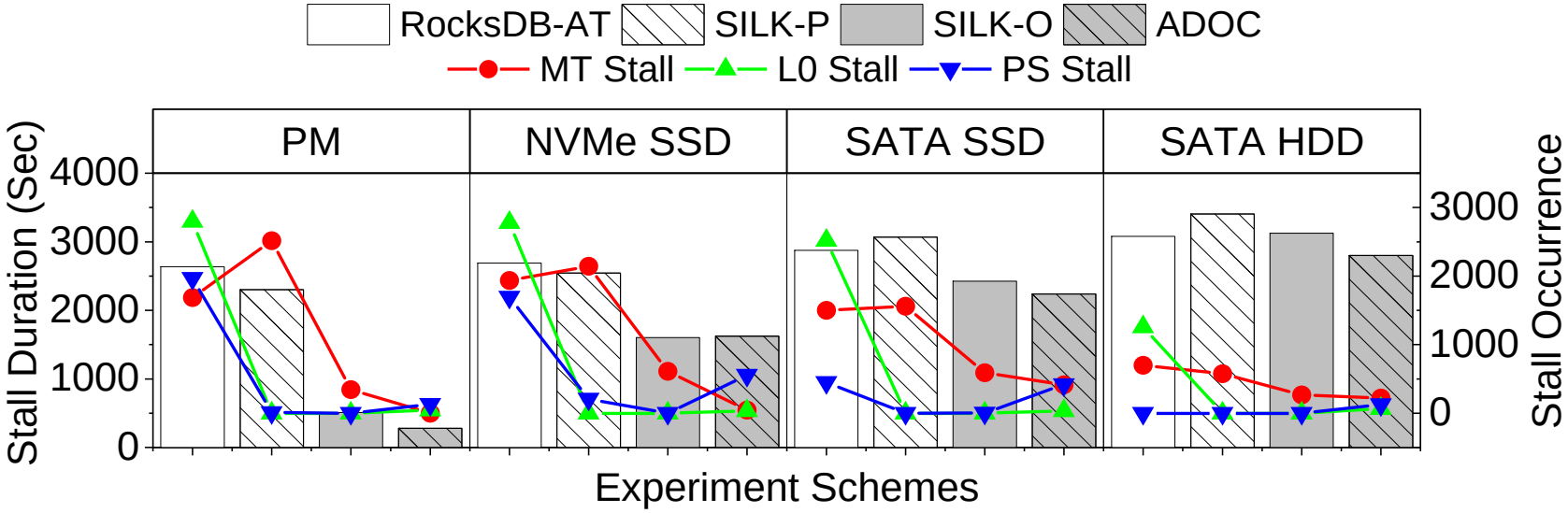
****SILK-O (SILK with 8 threads, 512 MB, manually optimized)**

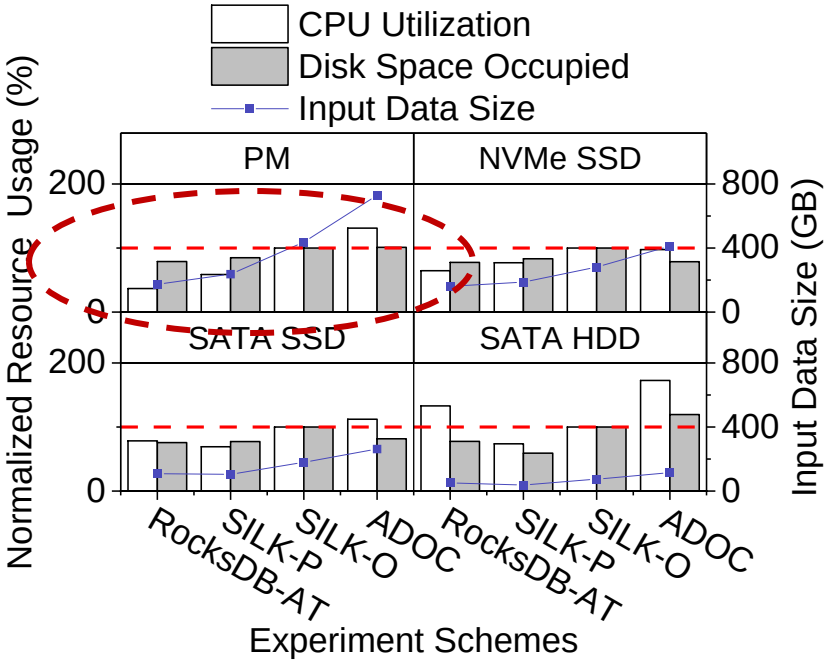
Throughput of *fillrandom* workload



- **Higher average throughput**
- **Higher peak throughput**
- **Lower stall duration and fewer stall occurrence**

Stall occurrences and resource usage

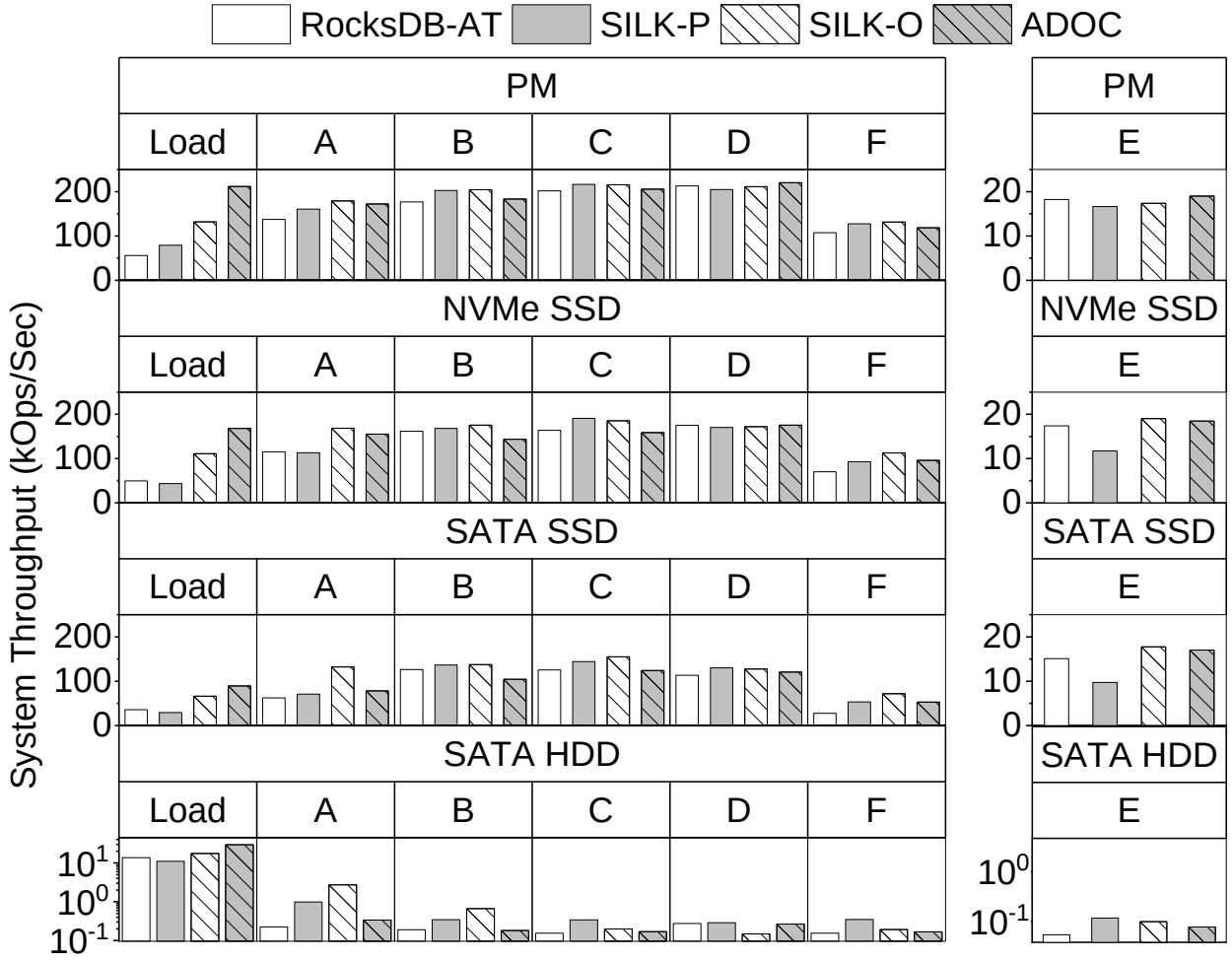




- **Stall reduction on all devices**
- **More input data, less space amplification**
  - + More compacted disk components
  - + Higher CPU utilization
  - - Slightly increase in occurrences of PS stall

Throughput of YCSB workloads

- **Lower improvements in loading stage**
  - Requests are in sequential order
  - Fewer write stalls occur
  - Fewer tuning operations
- **Slower than SILK-O in hybrid workloads**
  - Smaller writing pressure, fewer tuning operations
  - More data are flushed into disk components
  - Lower memory utilization



Conclusion

- **Understand underlying cause of write stalls**
  - Along ALL types of storage devices

➜ **Data Overflow: mismatch in data flow among LSM KV components**

- **Find solution to remove/alleviate write stalls**
  - For ALL types of storage devices

➜ **ADOC: Automatic Data Overflow Control**

Our code is available at https://github.com/supermt/FEAT_7.11

Thanks!
Questions?