HBaseConAsia2018

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

HBase and OpenTSDB Practices at Huawei

Pankaj Kumar, Zhongchaoqiang, Guoyijun, Zhiwei

{Pankaj.kr, zhongchaoqiang, guoyijun, wei.zhi}@huawei.com

$ whoami

HBase Tech Lead @ Huawei India
Apache HBase Contributor
5 years of experience in Big Data related projects

HBase @ Huawei

→ **Migrated from 1.0.2 version**

→ **1.3.1 version +**

- → Secondary index
- → MOB
- → Multi split

→ **Migrating to 2.1.x cluster this year**

Content

HBase Practices

- Accelerate HMaster Startup
- Enhanced Replication
- Reliable Region Assignment

1.1 Accelerate HMaster Startup

Accelerate HMaster Startup

Problem:

HMaster not available for longer duration on failover/restart

Deployment scenario:

- Large cluster with 500+ nodes
- 5000+ Tables and 120000 + regions
- 10 namespaces

Discovered problems in multiple areas in Master startups like below

- Slow region locality computation on startup
  - Serial region locality calculation
  - Too much time spent in region locality calculation
- HMaster aborting due to namespace initialization failure
  - Slow SSH/SCP
  - Similar to HBASE-14190
- Table info loading was taking too much time
  - High Namenode latency
  - Many other services creating lots of load in NN

Accelerate HMaster Startup

- **Slow region locality computation on startup**
  - ✓ Accelerate region Locality computation by computing in parallel
  - ✓ Detach region locality computation on startup
  - ✓ Similar solution HBASE-16570

- **HMaster aborting due to namespace initialization failure**
  - ✓ Assign system table regions ahead of user table regions
  - ✓ Assign system tables to HMaster (configure all system tables to hbase.balancer.tablesOnMaster)
  - ✓ On cluster/master startup, process old HMaster SSH/SCP ahead of other RegionServer
  - ✓ SSH/SCP will replay the WAL and assign the system table regions first

Accelerate HMaster Startup

Table Info Loading on Master startup

Example: Suppose there are two namespace and total 5 tables with below path structure in HDFS

Namespace	HDFS Path
default	t1/.tabledesc/.tableinfo.0000000001 t1/.tabledesc/.tableinfo.0000000002 t1/.tabledesc/.tableinfo.0000000003 t2/.tabledesc/.tableinfo.0000000001
hbase	/hbase/data/hbase/acl/.tabledesc/.tableinfo.0000000001 /hbase/data/hbase/meta/.tabledesc/.tableinfo.0000000001 /hbase/data/hbase/namespace/.tabledesc/.tableinfo.0000000001

Operation	Path	Result	Total RPC to NN
List operation to get path till all the namespace	/hbase/data/*	gets file status for all the namespaces	1
List operation on each namespace to get all the tables in each namespace	/hbase/data/default	get all the file status of all the tables in each namespace	2 (= total number of namespaces in the cluster)
List operation on each table to get all the tableinfo files for the table	/hbase/data/default/t1/. tabledesc	get all the file status of all the tableinfo files for the table	5 (= total number of tables in the cluster)
Open call for each table' s latest tableinfo file	/hbase/data/default/t1/. tabledesc/.tableinfo.0000000003	get the stream to tableinfo file	5 (= total number of tables in the cluster)
Total RPC			**13**

Total RPC to NameNode
=
1 + namespace count
+ 2 * table count

Accelerate HMaster Startup

Table Info Loading on Master startup

- 2011 RPC Calls (for 10 namespace and 1000 tables in a cluster)
- NN is busy then it will hugely impact the startup of HMaster.

Solution: Reduce number of RPC to Namenode

✓ HMaster makes a single call to get tableinfo path

- Get LocatedFileStatus of all tableinfo paths based on pattern (/hbase/data/*/*/.tabledesc/.tableinfo*)
- LocatedFileStatus will also contain block locations of tableinfo file along with FileStatus details
- DFS client will directly connect to Datanode through FileSystem#open() using LocatedFileStatus, avoid NN RPC to get the block location of the tableinfo file

Improvement:

In a highly overloaded HDFS cluster, it took around 97 seconds to load 5000 tables info as compared to 224 seconds earlier.

1.2 Enhanced Replication

Adaptive call timeout

Problem:

- Replication may timeout when peer cluster is not able to replicate the entries
- Can be solved by increasing *hbase.rpc.timeout* at source cluster
  - Impact other RPC request
  - In Bulkload replication scenario fixed RPC timeout may not guarantee bigger HFile copy
- Refer **HBASE-14937**

Solution:

- ✓ Source cluster should wait longer
- ✓ New configuration parameter *hbase.replication.rpc.timeout*, default value will be same as *hbase.rpc.timeout*
- ✓ On each *CallTimeOutException* increase this replication timeout value by fixed multiplier
- ✓ Increase the replication to certain number of configured times

Cross realm support

Problem:

- Replication doesn’ t work with Kerberos Cross Real Trust where principal domain name is not machine hostname
- On new host addition
  - Add principal name for the newly added hosts in KDC
  - Generate a new keytab file
  - Update it across other hosts
- Rigorous task for user to create and replace new keytab file

Solution:

- HBASE- 14866
  - ✓ Configure the peer cluster principal in replication peer config
- Refer to HBASE-15254 (Open)
  - ✓ No need to configure in advance, fetch at runtime.
  - ✓ Make RPC call to peer HBase cluster and fetch the Principal
  - ✓ Make RPC connection based on this server principal

1.3 Reliable Region Assignment

RIT Problem

Problem:

- Region stuck in transition for longer duration due to some fault in cluster
  - Zookeeper node version mismatch
  - Slow RS response
  - Unstable Network
- Client will not be able to perform read/write operation on those regions which are in transition
- Balancer will not run
- Region can’ t be recovered until cluster restart

Solution:

- ✓ Recover the regions by reassigning them
- ✓ Schedule a chore service
  - Run periodically and identify the region which stuck in transition from a longer duration (configurable threshold)
  - Recover the region by reassigning them
- ✓ New HBCK command to recover regions which are in transition from longer duration

Double Assignment Problem

Problem:

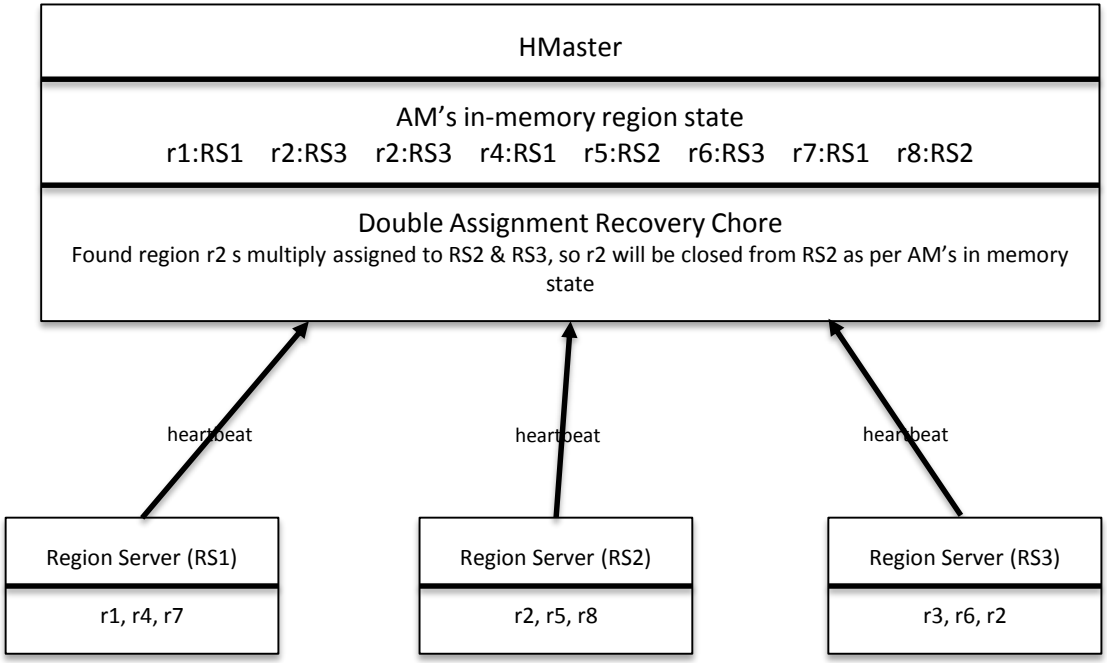
- HMaster may assign region to multiple RegionServer in a faulty cluster environment
  - Call time out from a overloaded RegionServer
- Old or new client may receive inconsistent data
- Can’ t be recovered until cluster restart

Solution:

- ✓ Multiply assigned regions should be closed and assign uniquely
- ✓ Region server send server load details to HMaster through heartbeat
- ✓ Schedule a chore service which run periodically and recover the regions
  - Collect each region server load from HMaster memory
  - Identify the duplicate regions from the region list
  - Validate the duplicate regions with HMaster Assignment Manager in-memory region state
  - Close the region from the old region server
  - Assign the region

Double Assignment Problem

Example:



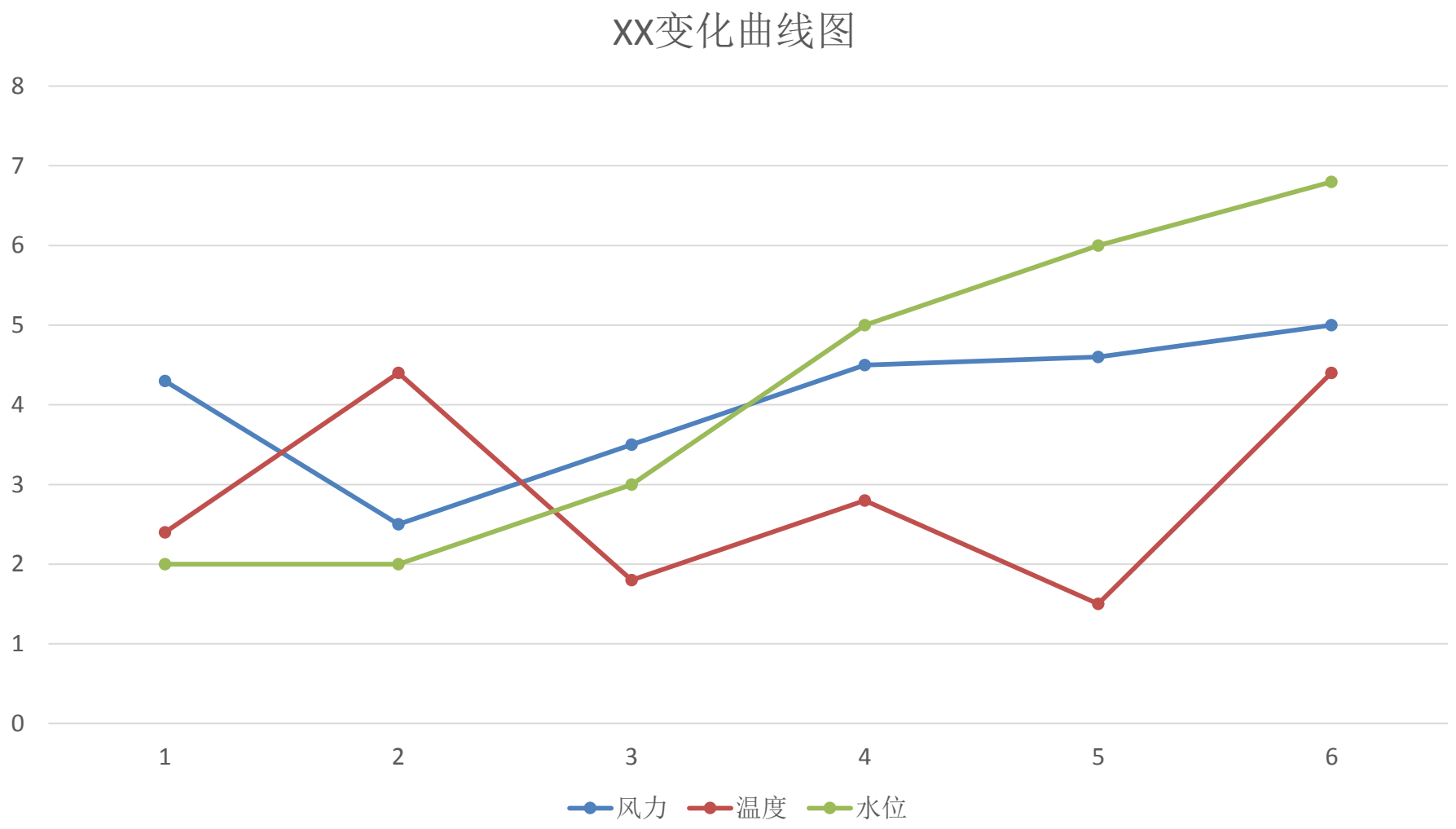
OpenTSDB Improvement

OpenTSDB Basics

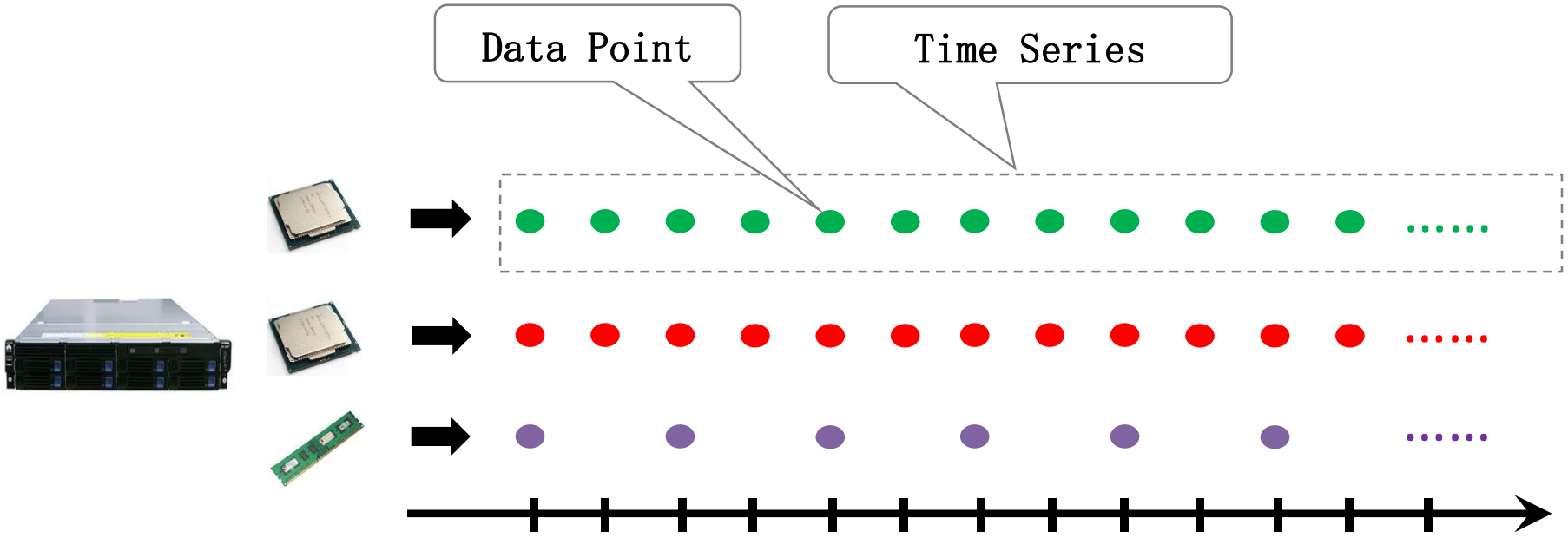
OpenTSDB Improvement

2.1 TSDB Basics

Time Series



Time Series

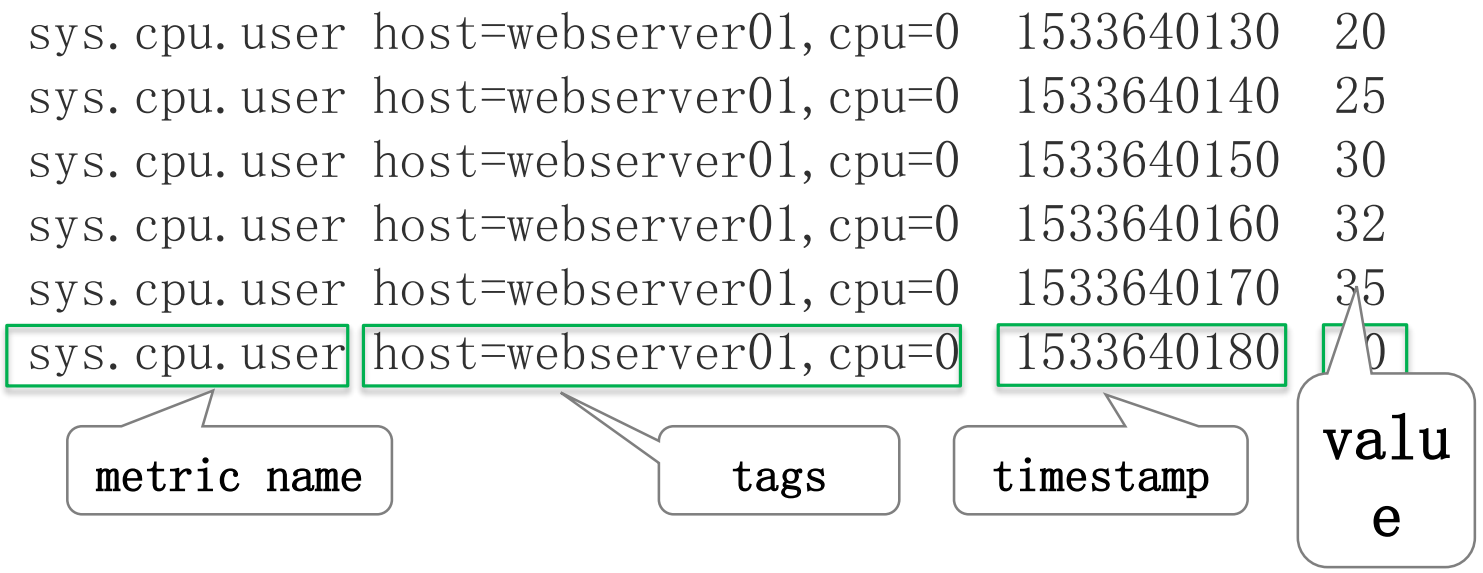


A **time series** is a series of numeric **data points** of some particular **metric** over time.

- OpenTSDB Document

HUAWEI

OpenTSDB Schema



OpenTSDB uses a **metric name** and a group of **tags** for identifying time series. **Tags** are used to identify different data sources.

TSDB Characteristics

- **Write Dominate**. Read rate is usually a couple orders of magnitude lower.
- Most queries happens on **latest data**.
- Most queries are for **aggregate analysis** instead of individual data point.
- **Primarily Inserts**. Updates/deletions are rarely happens.

Basic Functionality For TSDB

- Rollups and Downsampling
- Pre-aggregates and Aggregates
- Interpolation
- Data Life Cycle Management

Single value model vs. Multi-value model

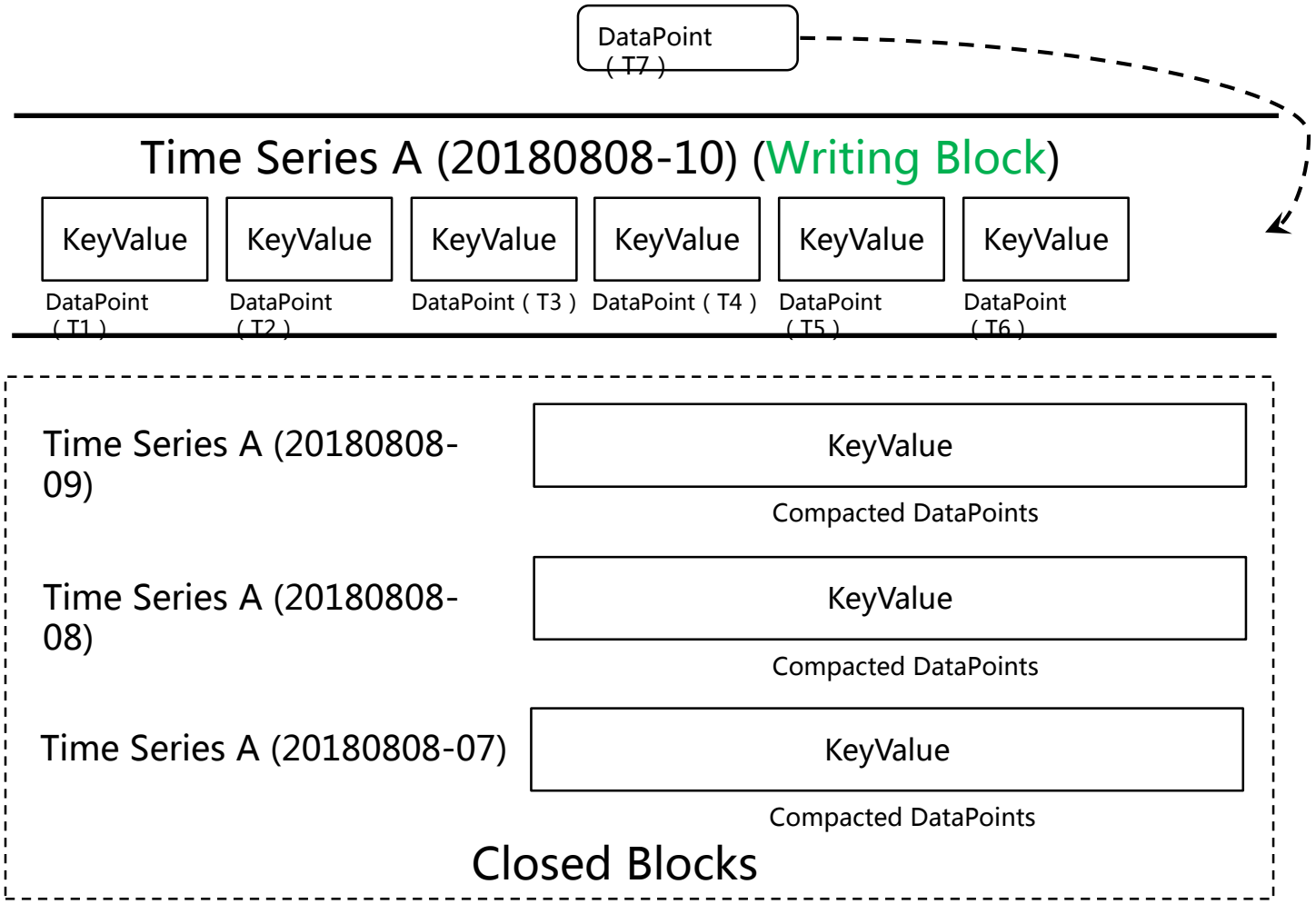
Metric	TimeStamp	DeviceID	DeviceType	ZoneId	Temperature	Pressure	WaterLine
Engine	20180101 12:00:00	ID001	TypeA	1	66.9	1.33	42.5
	20180101 12:00:00	ID002	TypeA	1	68.8	1.28	42.0
	20180101 12:00:00	ID003	TypeA	1	67.3	1.35	41.7
	20180101 12:01:00	ID001	TypeA	1	67.5	1.30	42.2

Metric Time stamp Tags Field

Metric	TimeStamp	DeviceID	DeviceType	ZoneId	Value
Temperature	20180101 12:00:00	ID001	TypeA	1	66.9
Pressure	20180101 12:00:00	ID001	TypeA	1	1.33
WaterLine	20180101 12:00:00	ID001	TypeA	1	42.5
Temperature	20180101 12:01:00	ID002	TypeA	1	68.8
Pressure	20180101 12:01:00	ID002	TypeA	1	1.28
WaterLine	20180101 12:01:00	ID002	TypeA	1	42.0

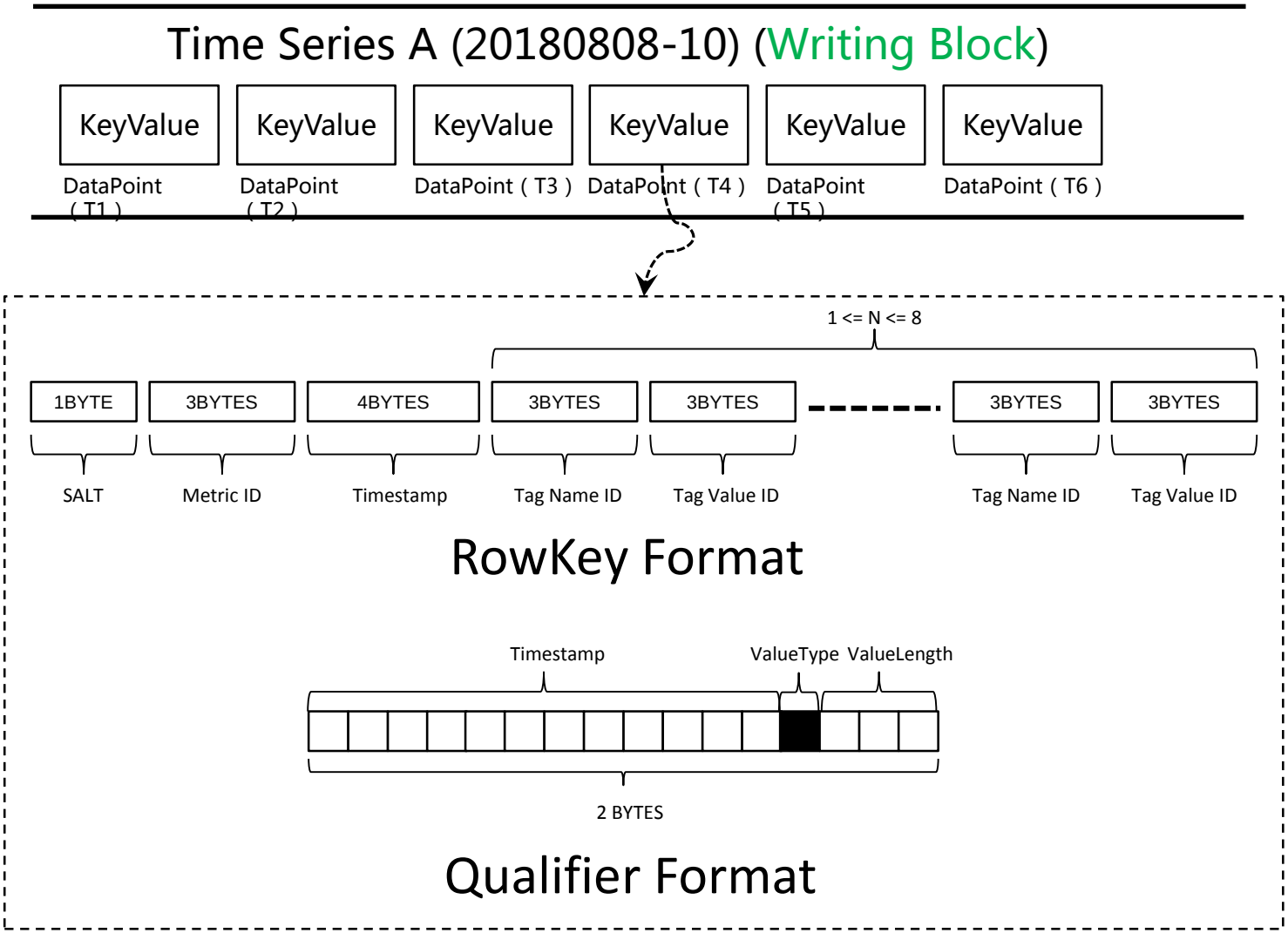
Metric Time stamp Tags Metric Value

Time Series Storage In HBase



Time Series Separeted into multiple blocks. Each block hold one hour of data points.

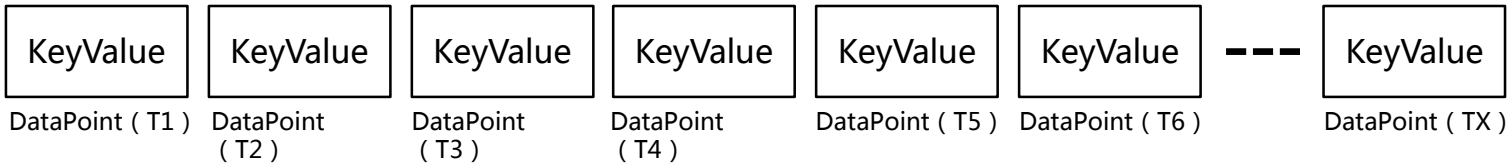
OpenTSDB Table Design



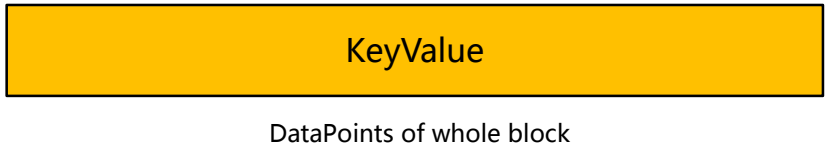
HUAWEI

OpenTSDB Compaction

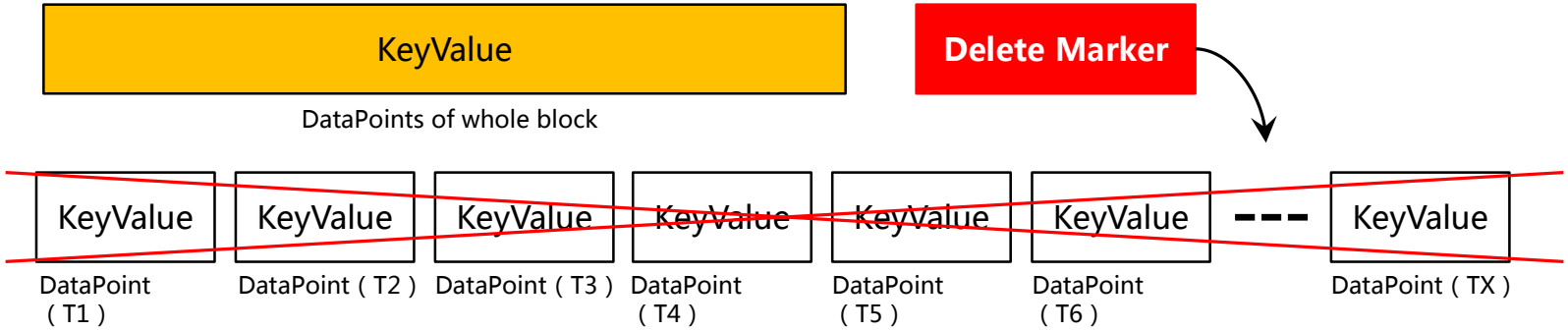
1. Read All Data Points from the block of Last Hour



2. Compact locally



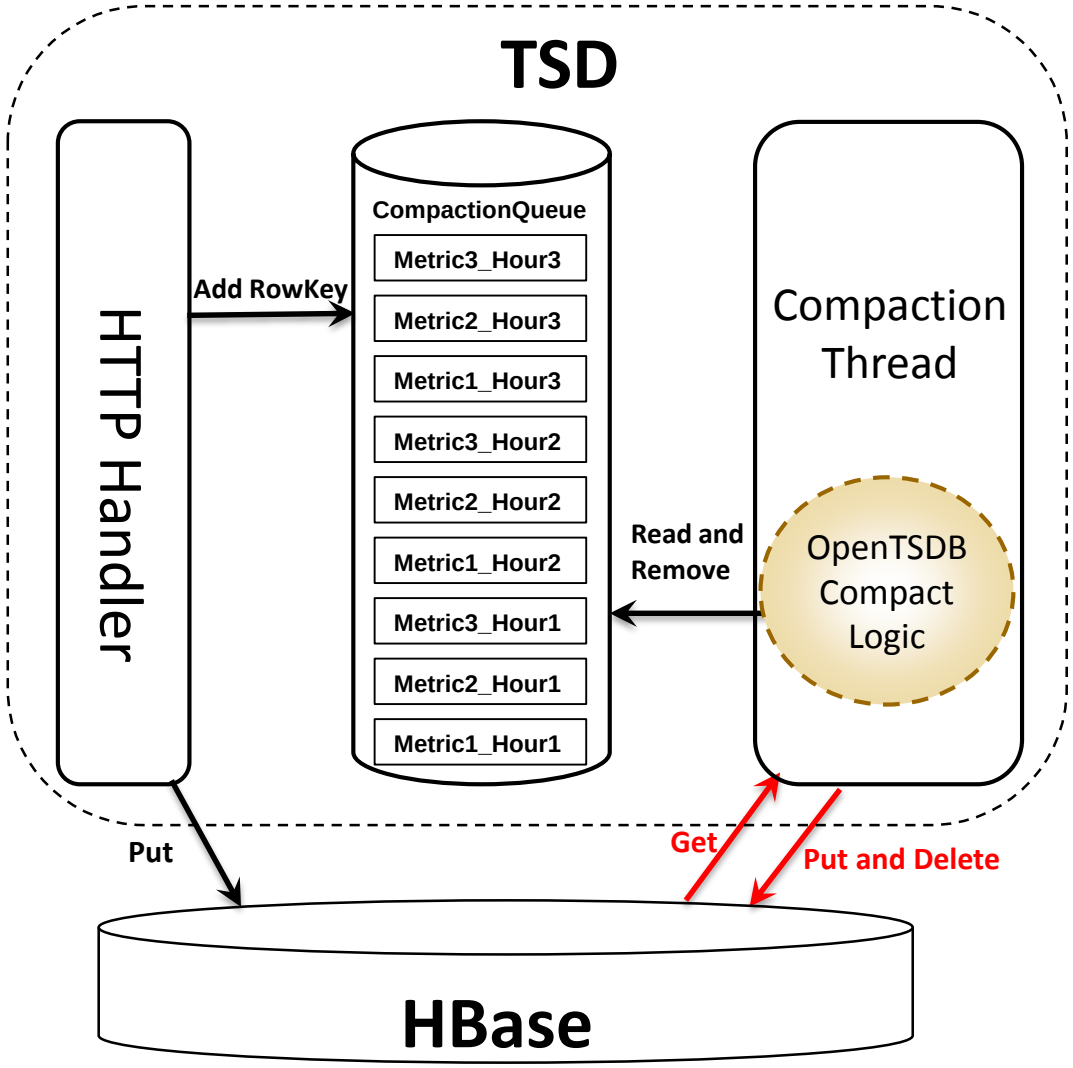
3. Write compact row, and delete all exist individual data points



HUAWEI

2.1 OpenTSDB Improvement

Exist OpenTSDB Compaction Flow

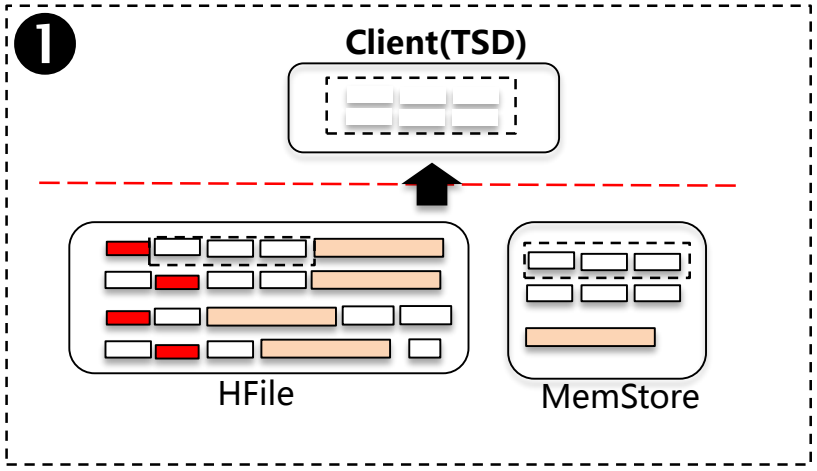


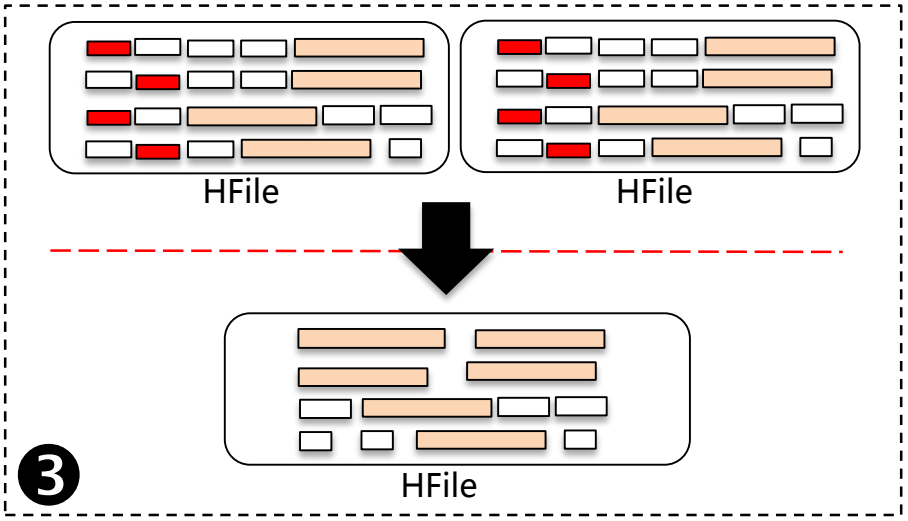
OpenTSDB compaction is helpful for read, and could decrease total data amount , but the side effects as follows:

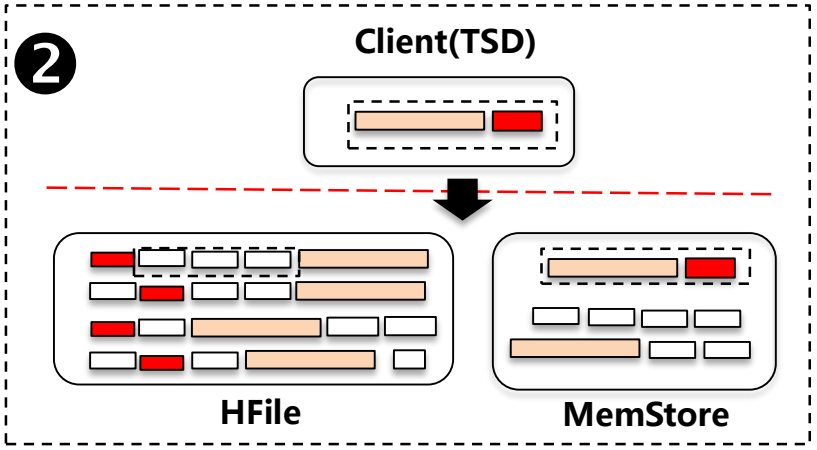
1. OpenTSDB Compaction requires a read/compact/write cycle, causing extremely high traffic to RegionServers.

2. Write compact row and delete exist individual data points **amplify** write I/O.

Understanding Write Amplification



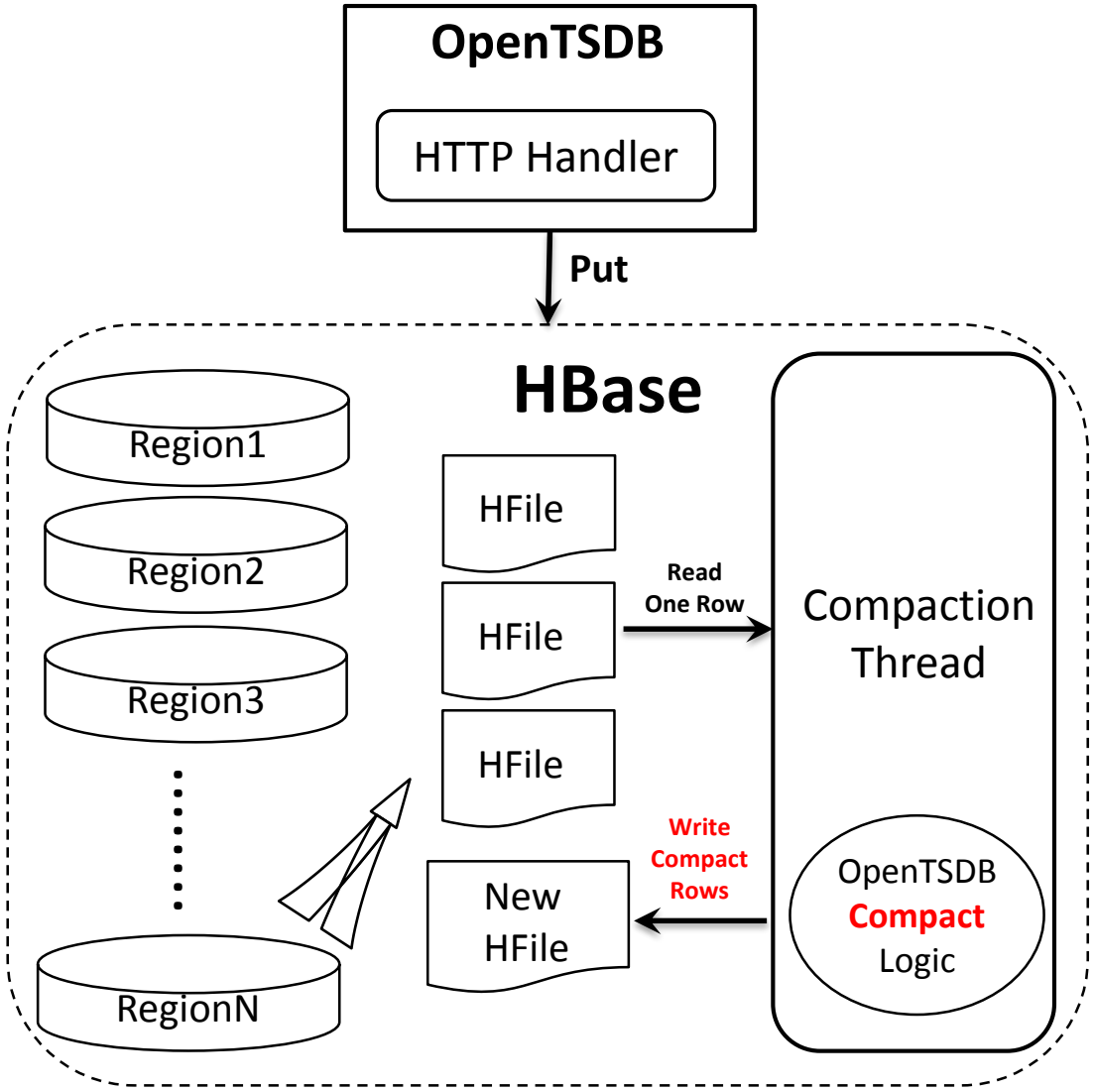




1. TSD client read time series data from Regionserver.
2. TSD write compacted row and delete marker to RegionServer.
3. HBase internal compaction.

Single Datapoint | Delete Marker | Compact Row

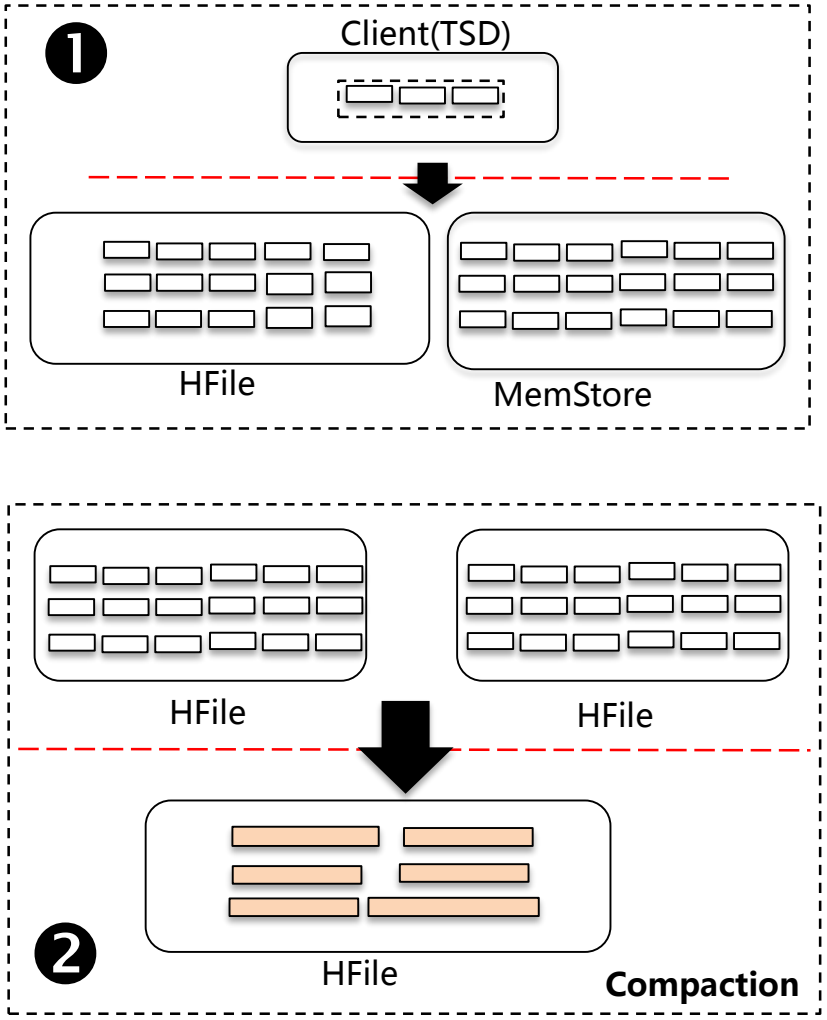
New OpenTSDB Compaction Flow



New HBase Compaction implementation for OpenTSDB that could compact data points during hbase compaction running.

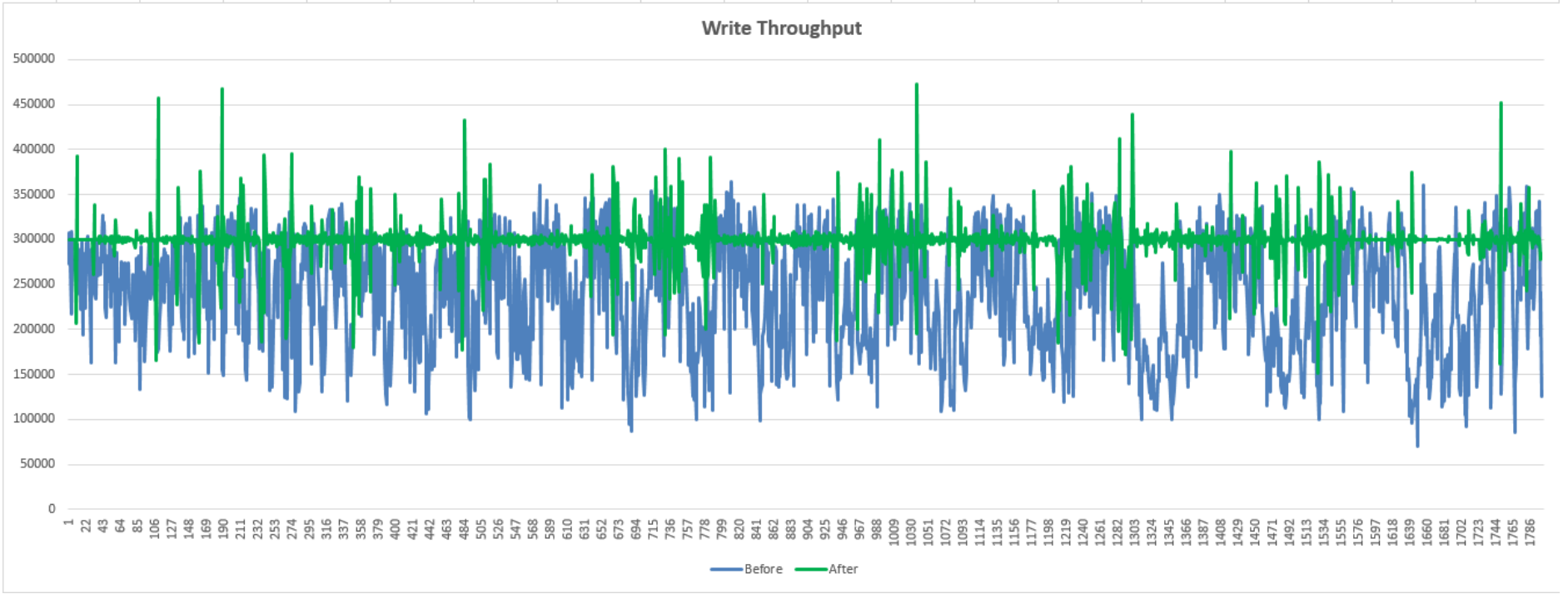
Making TSD focus on handing user read/write requests only.

No More Extra Write Amplification



No more extra write amplication caused by OpenTSDB data points compaction.

Benchmark – Throughput comparison

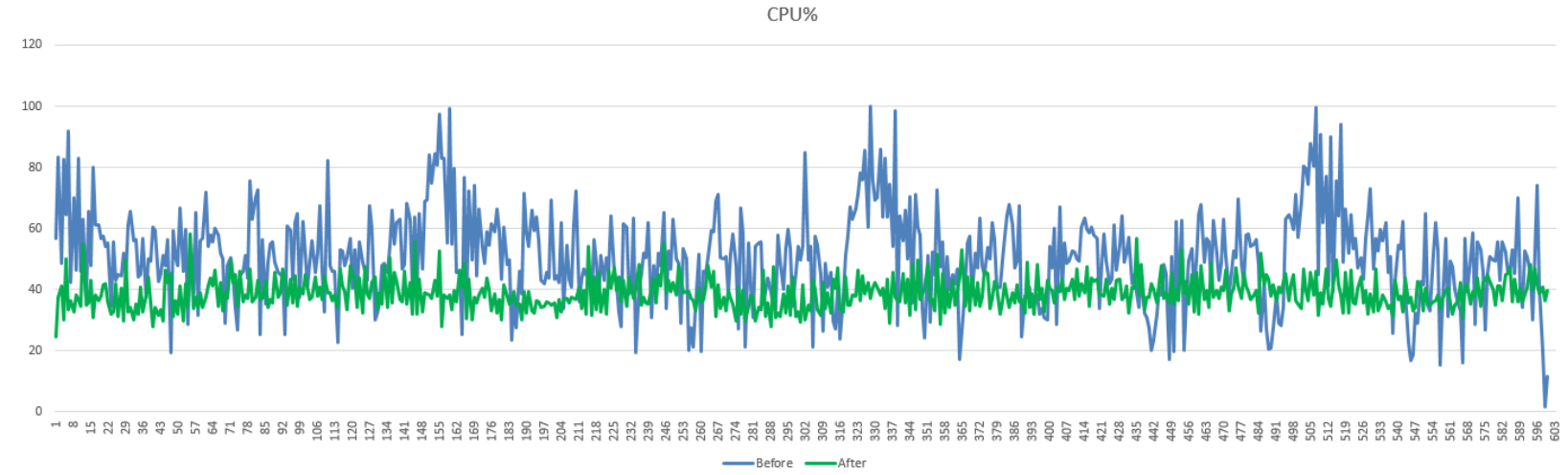


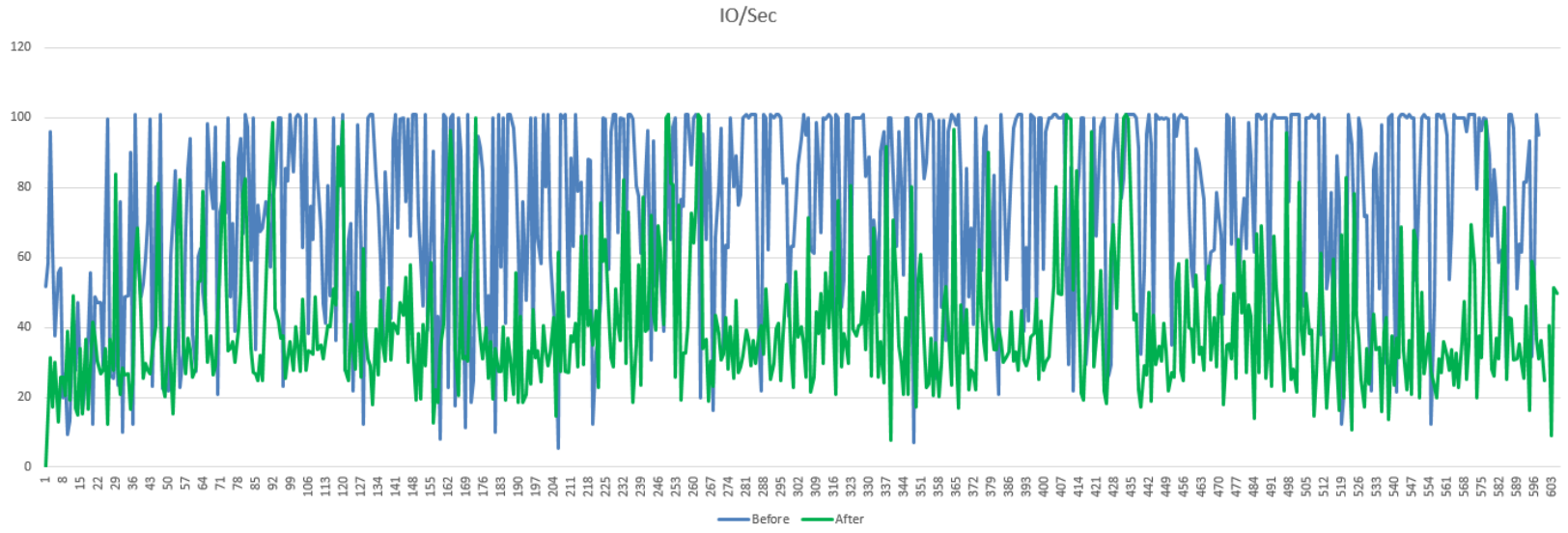
NOTE: TSDs were limited to 300,000 data points per second.

After optimization, write throughput has been improved significantly.

HUAWEI

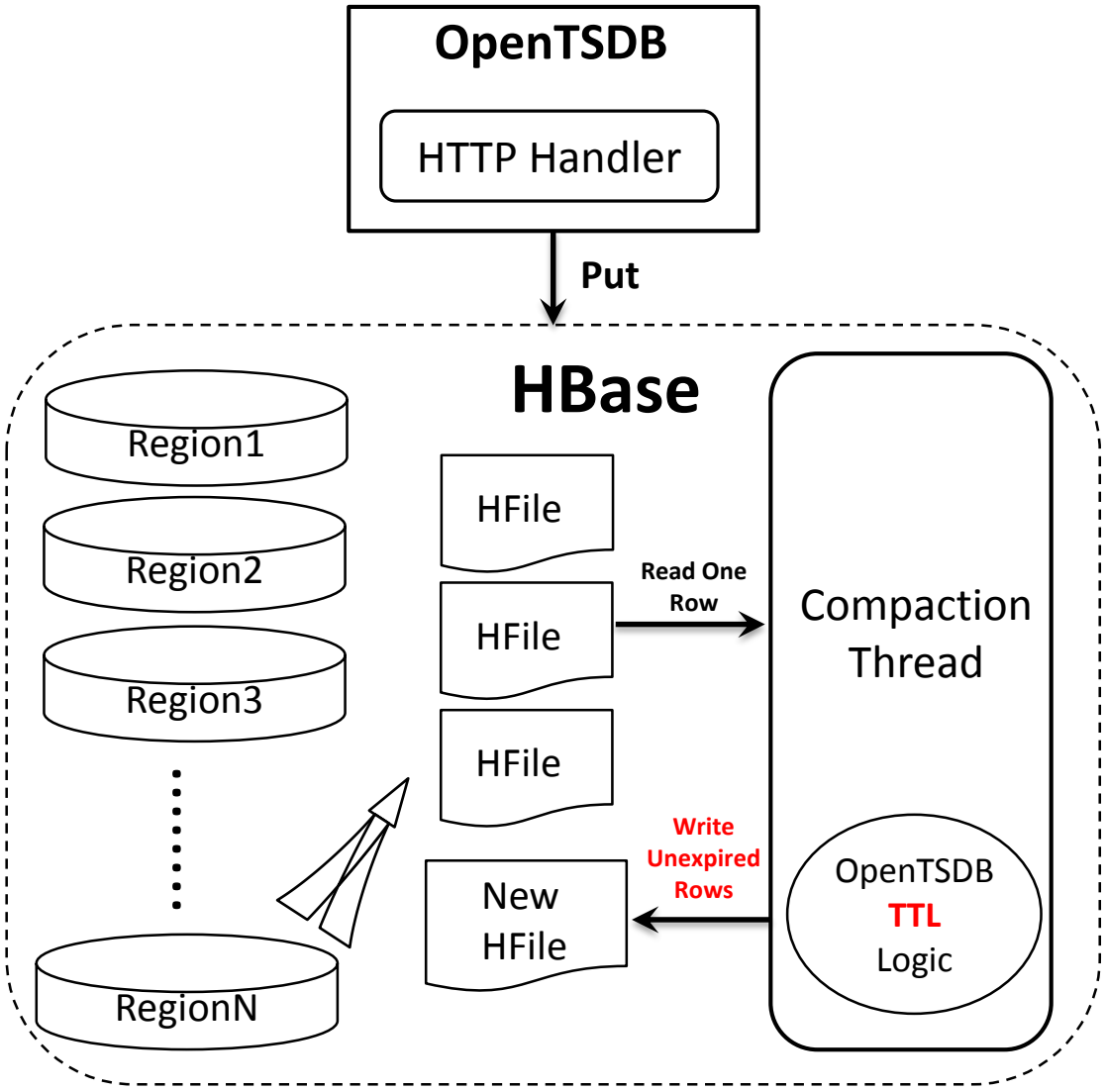
Benchmark – CPU And IO Comparison





HUAWEI

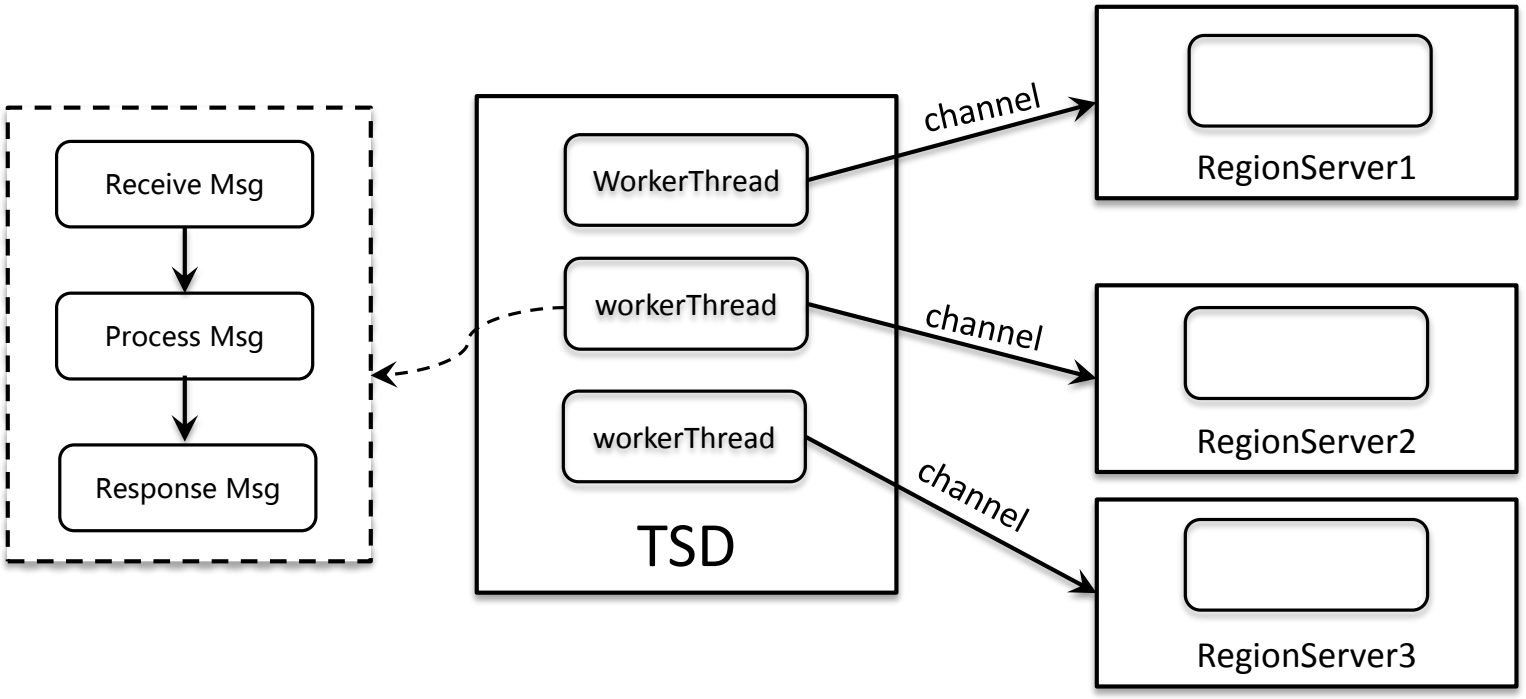
Data Life Cycle Management Per Metric



- Delete old data automatically for reduce data load.
- HBase Table level TTL is a coarse-grained mechanism, But different metrics may have different TTL requirements.
- A new HBase compaction implementation for per-metric level data life cycle management.

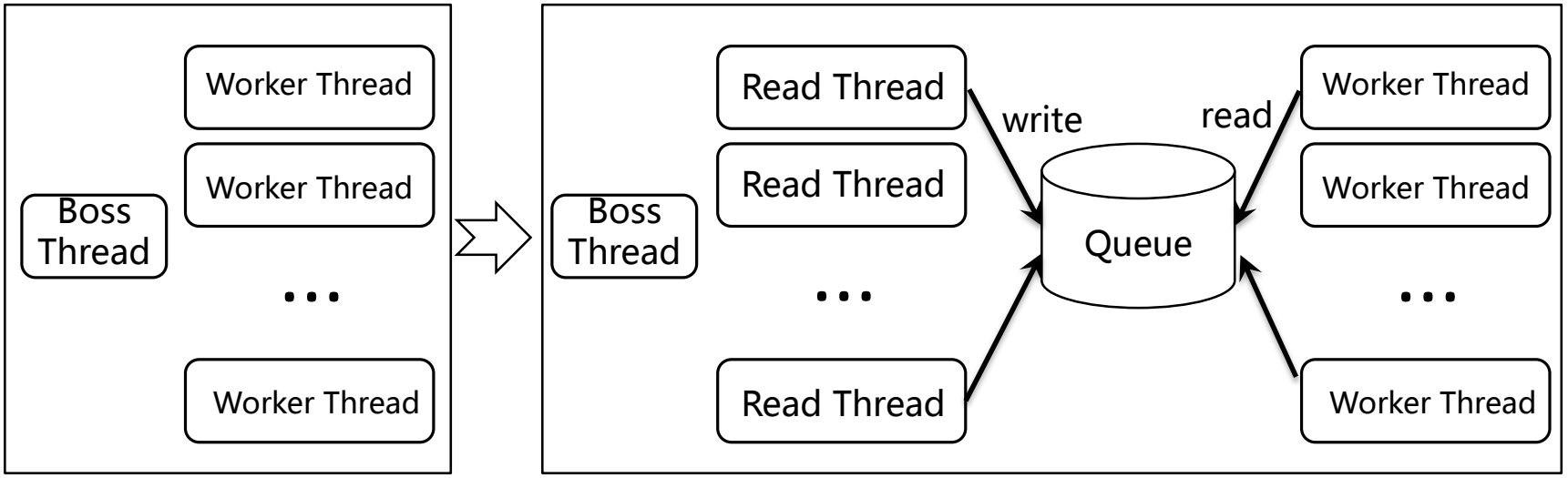
OpenTSDB RPC Thread Model

Using a two-level thread model design. Receive message, process message and response to client are all handled by one WorkThread. It causes the low CPU usage.



RPC Thread Model Improvement

Modify the thread model to a Three-Level design. Receiving message and handling message are finished in different threads. Better CPU usage.



Benchmark:

Query latency got at least **3X** improvement for concurrent queries.

	Before	After
1 Query	60ms	59ms
10 Concurrent Queries	476ms	135ms
50 Concurrent Queries	2356ms	680ms

Follow Us（关注我们）

华为云CloudTable

公有云HBase服务

微信公众号

NoSQL漫谈

HUAWEI

Thank You.