Experiences with Building High-Performance Distrusted Block Storage at SmartX

张凯 CTO at SmartX

MAKE IT SIMPLE

Agenda

- Background & challenges
- Architecture of SmartX ZBS
- Experiences with SmartX ZBS
- Roadmap

About SmartX

- Founded by three *geeks* in 2013
- Focused on *distributed systems* and *virtualization*
- *Leading* Chinese HCI technology and market
- Product has been deployed on *thousands* of hosts and running for 2 years
- Cooperate with Chinese *leading* hardware venders and cloud venders
- Has now raised *10s of millions of dollars*

About Me

- Computer Science, Tsinghua University
- Software Engineer, Infrastructure Team, Baidu
- Cofounder & CTO, SmartX
- Contributor, Sheepdog

Agenda

- Background & challenges
- Architecture of SmartX ZBS
- Experiences with SmartX ZBS
- Roadmap

Why Block Storage Matters

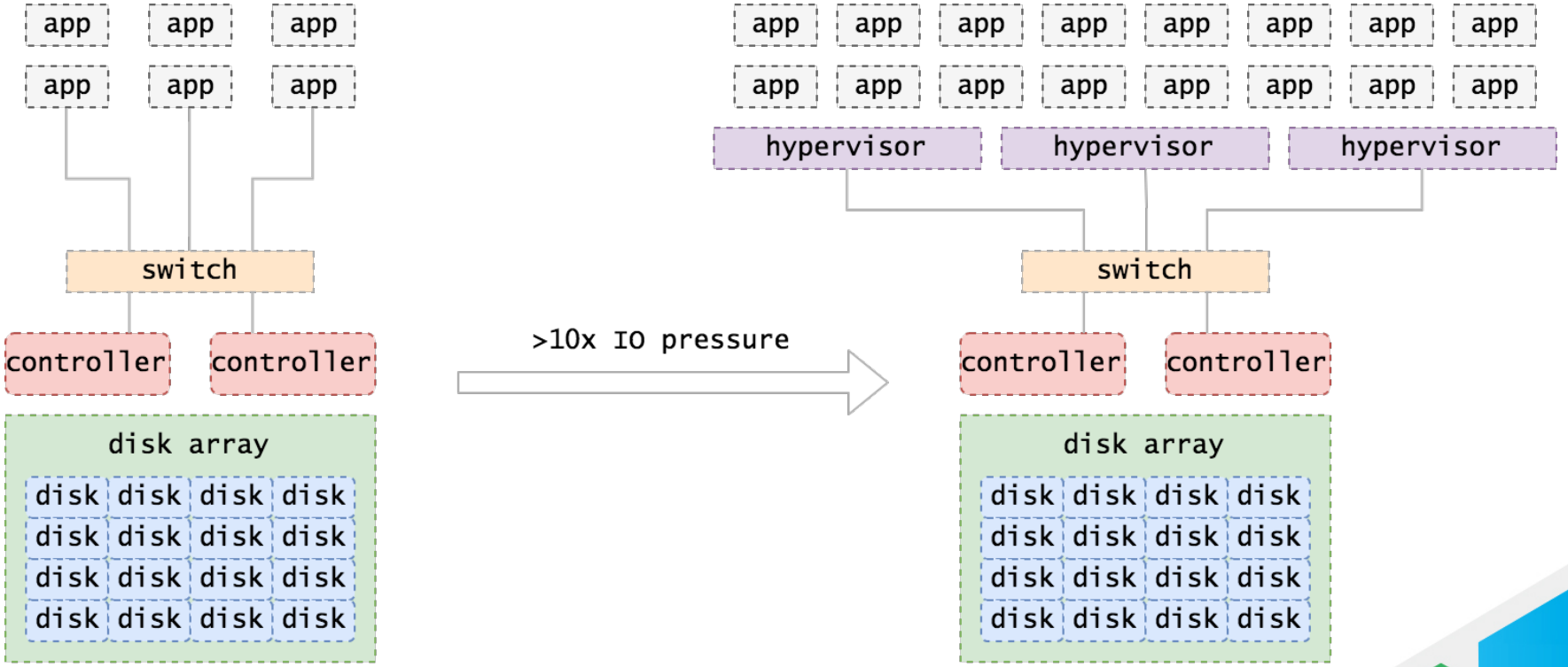
- Different kinds of storage
  - Block: used by hypervisor like esxi, kvm and xen
  - File: HPC and shared workspace
  - Object: store small files like image and voice
- Block storage stores most valuable data
  - Applications: database, mail server, VDI, AD, …
  - Industries: finance, bank, manufacture, hospital, …
- The value of storage system is determined by the value of data it stores

Why Block Storage Matters

smartx
MAKE IT SIMPLE

- Highest resiliency requirement
  - No single point of failure, self healing
  - Data checksum, efficient snapshot, disaster recovery
- Highest performance requirement
  - Normally less than 1ms response time, more than 1GBps bandwidth, 5k IOPS per database instance
  - No performance cliff
- Scalability & agility
  - Deployment, scaling, shipment

Traditional Storage Sucks in Cloud



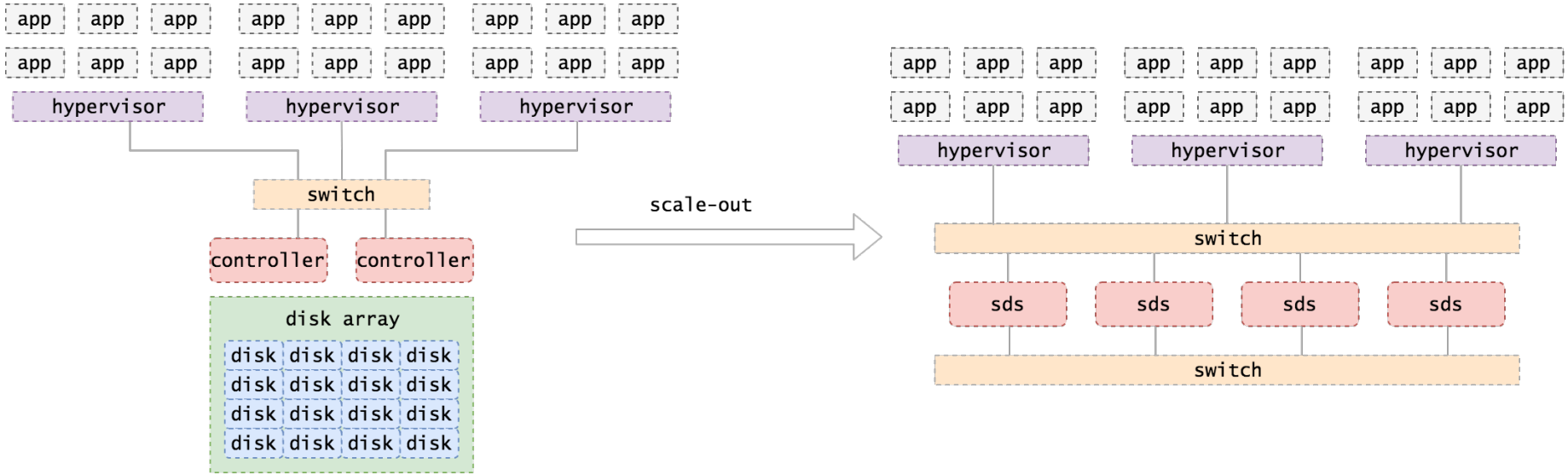
Traditional Storage Sucks in Cloud

smartx
MAKE IT SIMPLE

- Pros of traditional storage
  - Run well in small size
  - Mature product
- Cons of traditional storage
  - Scale-up architecture, not scale-out
  - Customized hardware, too long to acquire and hard to deployment
  - Expensive

We Need Cloud-Native Storage

smartx
MAKE IT SIMPLE

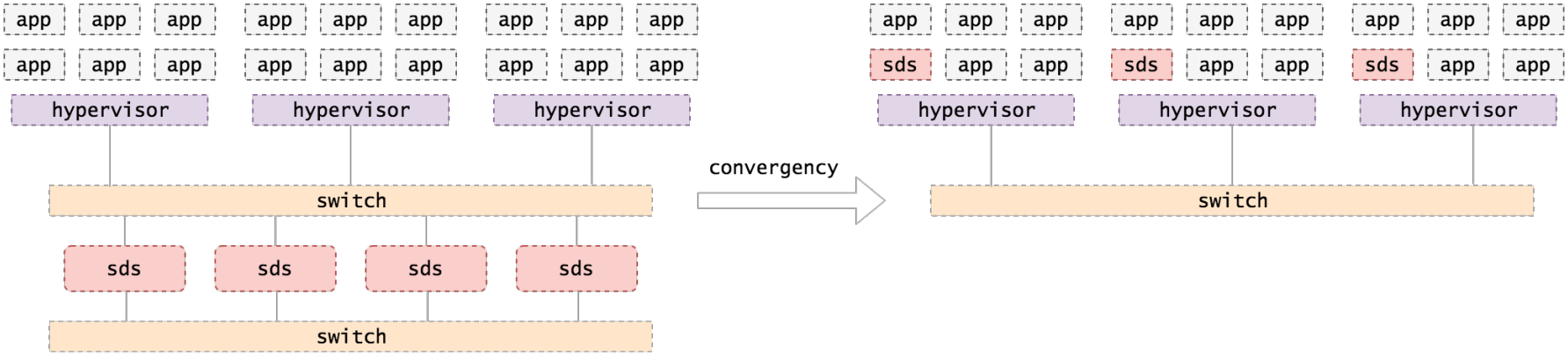


We Need Cloud-Native Storage

smartx
MAKE IT SIMPLE

- Pros of SDS
  - Scale-out
  - Commodity hardware
- Cons of SDS
  - Hard to management

Hyper-Converged Infrastructure



The core technology of HCI is SDS

Hyper-Converged Infrastructure

smartx
MAKE IT SIMPLE

- Avoid network latency by accessing data locally
- Unified compute node and storage node
- Less space and power consumption
- Easy to management
  - Appliance with VM management, SDS, SDN, hardware management …

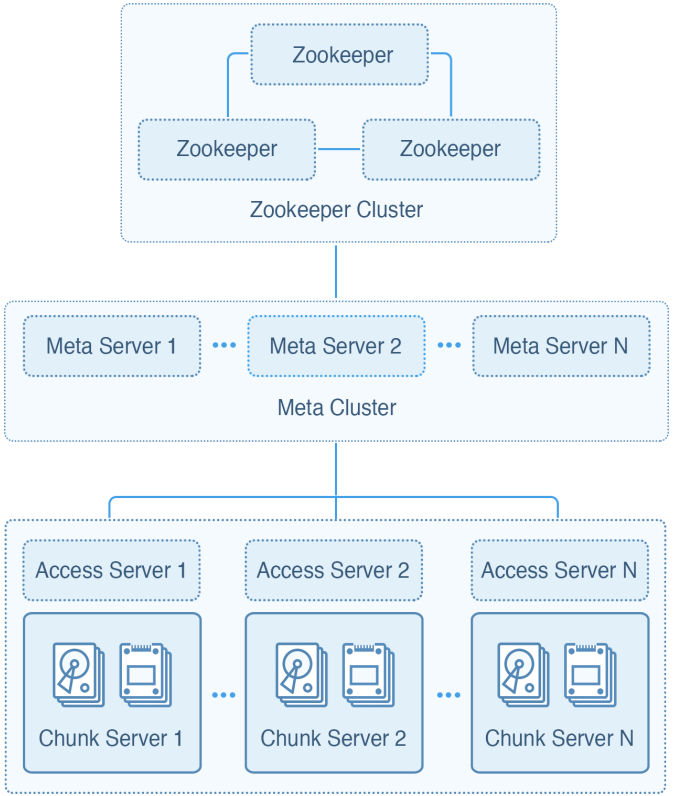
HCI Challenges

- Limited resources
  - About 10% of host
  - Typically: **4 vCPU** and **8GB memory** for SDS
- Commodity hardware
  - Not reliable as customized hardware
  - No NVRAM
  - No RAID controller

Agenda

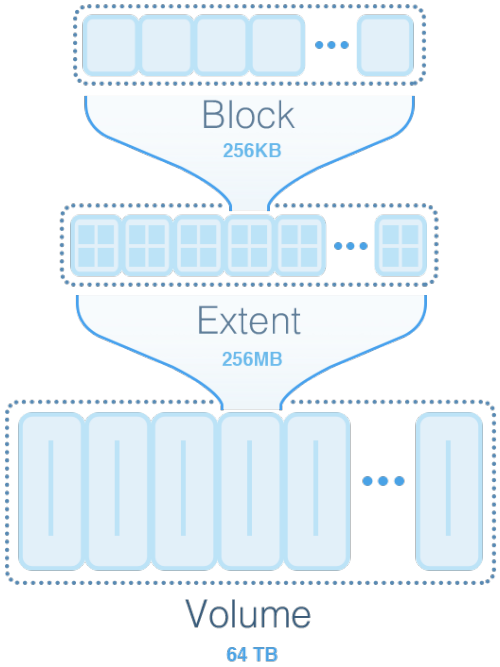
- Background & challenges
- Architecture of SmartX ZBS
- Experiences with SmartX ZBS
- Roadmap

Architecture of SmartX ZBS



- Zookeeper:
  - Cluster consistency
- Meta Server:
  - Meta data management
  - LevelDB + Zookeeper
- Access Server
  - iSCSI & NFS gateway
  - High availability
- Chunk Server
  - Userspace filesystem
  - Local data consistency

ZBS Data Structure



- Provision unit of Chunk Server
- Provision unit of Meta Server
- Storage policy container:
  - Replication factor
  - Thin provision
  - Access control

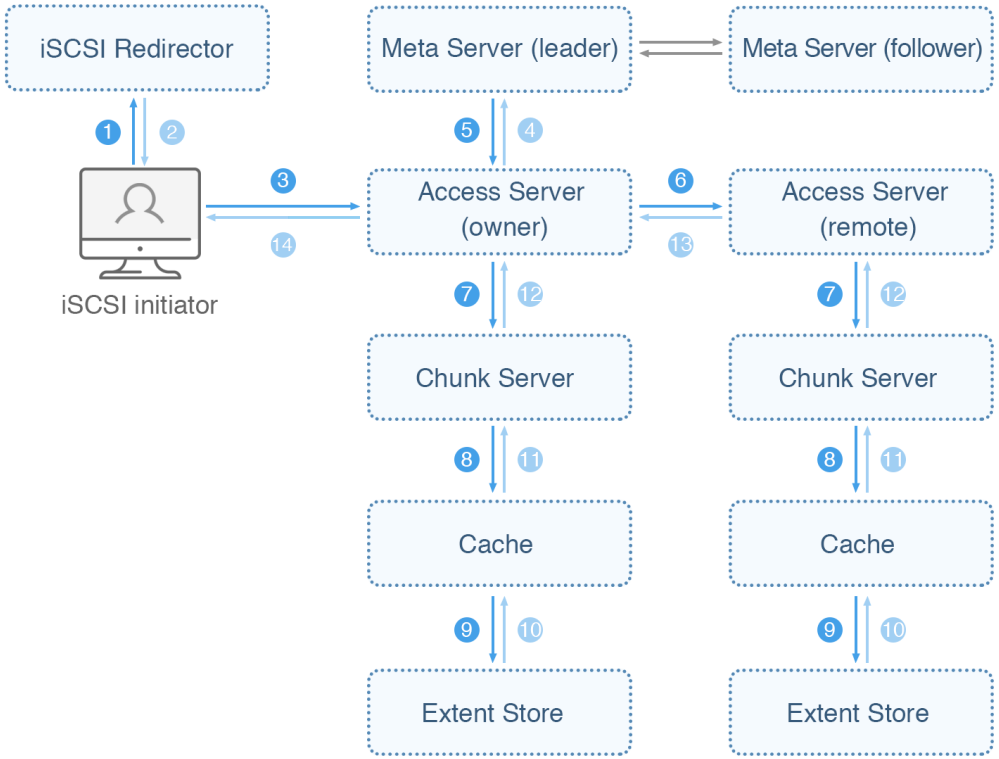
ZBS Highlight Features

- High efficiency
  - 2 vcpu and 4GB memory for Access Server and Chunk Server
  - 2 vcpu and 4GB memory for Meta Server, Zookeeper and others
  - Better performance than Ceph with same hardware while 10 times resource consumption less than Ceph
- Data protection
  - Replication
  - Snapshot, clone
  - Metro-availability
  - Data checksum

ZBS Highlight Features

- iSCSI/NFS gateway
  - Accessible on every node
  - Access locally by default
  - Reroute to remote when local failure
- Open ecosystem
  - Support esxi, kvm, xen and container
  - Support VMware VAAI, Citrix Ready, Openstack, Kubernetes

Data Flow (iSCSI)



Agenda

- Background & challenges
- Architecture of SmartX ZBS
- Experiences with SmartX ZBS
- Roadmap

Experiences with SmartX ZBS

- IO performance optimization

- Asynchronous programming framework with coroutine

About Disk IO Performance

smartx
MAKE IT SIMPLE

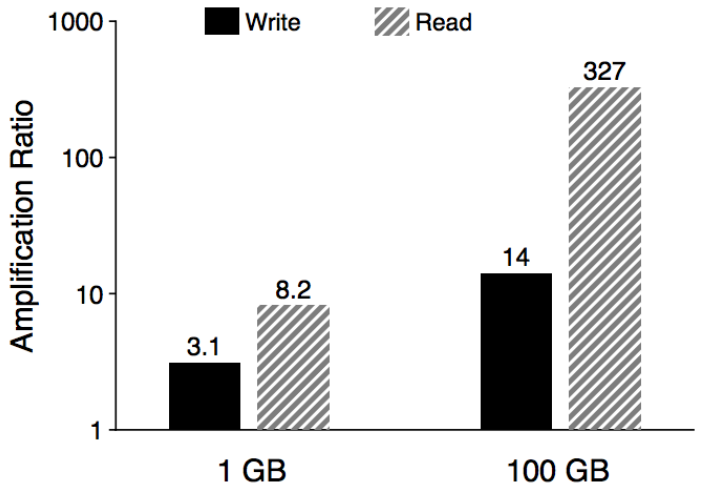
- Reduce write amplification

- Explore flash parallelism

Write Amplification

- Logs on logs on logs …
  - LevelDB/RocksDB: write ahead log
  - Ext4: journaling
  - FTL: log structured

How Much Write Amplified



Leveldb

https://www.usenix.org/system/files/conference/fast16/fast16-papers-lu.pdf

Measure	ext2	ext4	xfs	f2fs	btrfs
File Overwrite					
Write Amplification	2.00	4.00	2.00	2.66	32.65
Space Amplification	1.00	4.00	2.00	2.66	31.17
File Append					
Write Amplification	3.00	6.00	2.01	2.66	30.85
Space Amplification	1.00	6.00	2.00	2.66	29.77
File Read (cold cache)					
Read Amplification	6.00	6.00	8.00	9.00	13.00
File Read (warm cache)					
Read Amplification	2.00	2.00	5.00	3.00	8.00

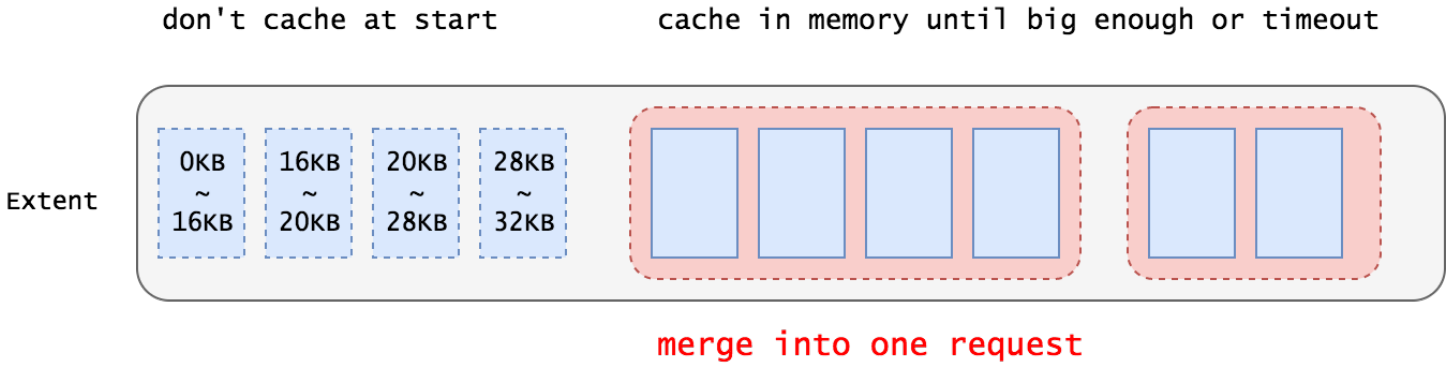
Filesystems

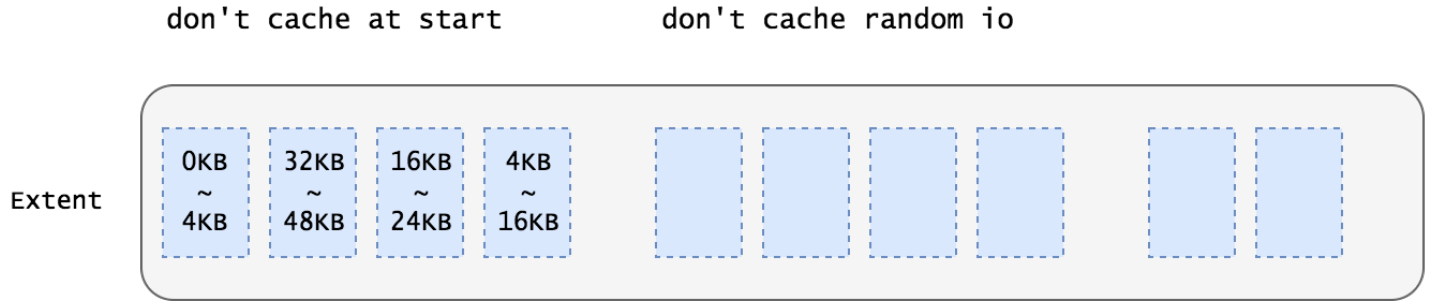
https://arxiv.org/pdf/1707.08514.pdf

How ZBS reduce write amplification

- Don’t use LevelDB/RocksDB for data
- Bypass kernel filesystem
  - Avoid performance overhead
  - Manage multiple devices in userspace
  - Use flash as cache and journal
- Don’t journal data for sequential write
  - Allocate new blocks

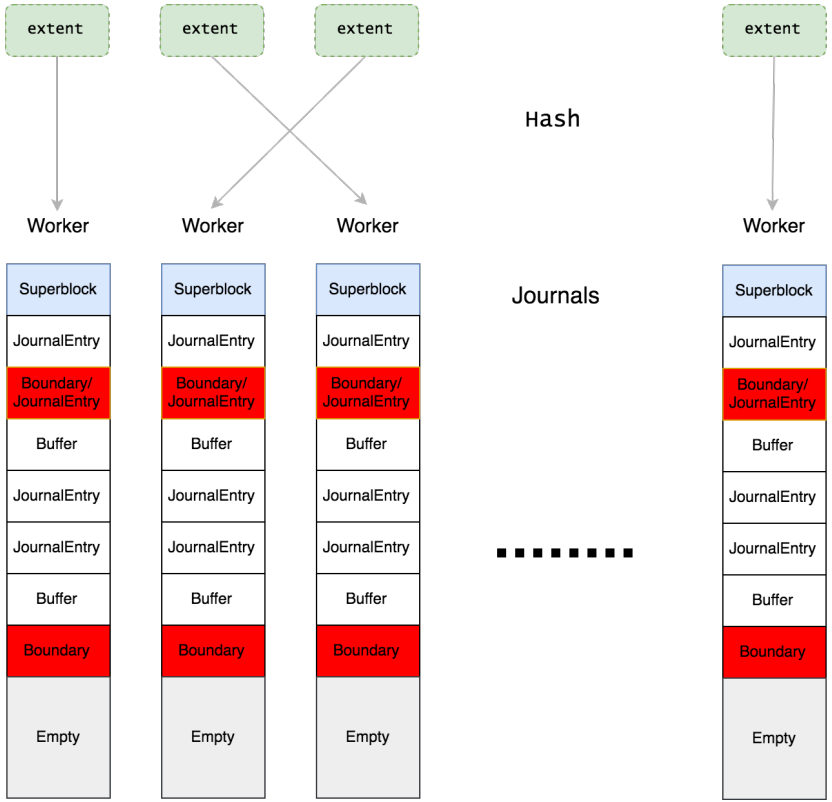
IO Pattern Recognization





Multiple Journals

- Extents are hashed into workers
- Each worker has an individual journal



Experiences with SmartX ZBS

- IO performance optimization

- Asynchronous programming framework with coroutine

What is Coroutine

smartx
MAKE IT SIMPLE

- Each coroutine has an individual context
- Non-preemptive
- One thread may run many coroutines
- Write asynchronous program in synchronous way

RPC Example with Coroutine

smartx
MAKE IT SIMPLE

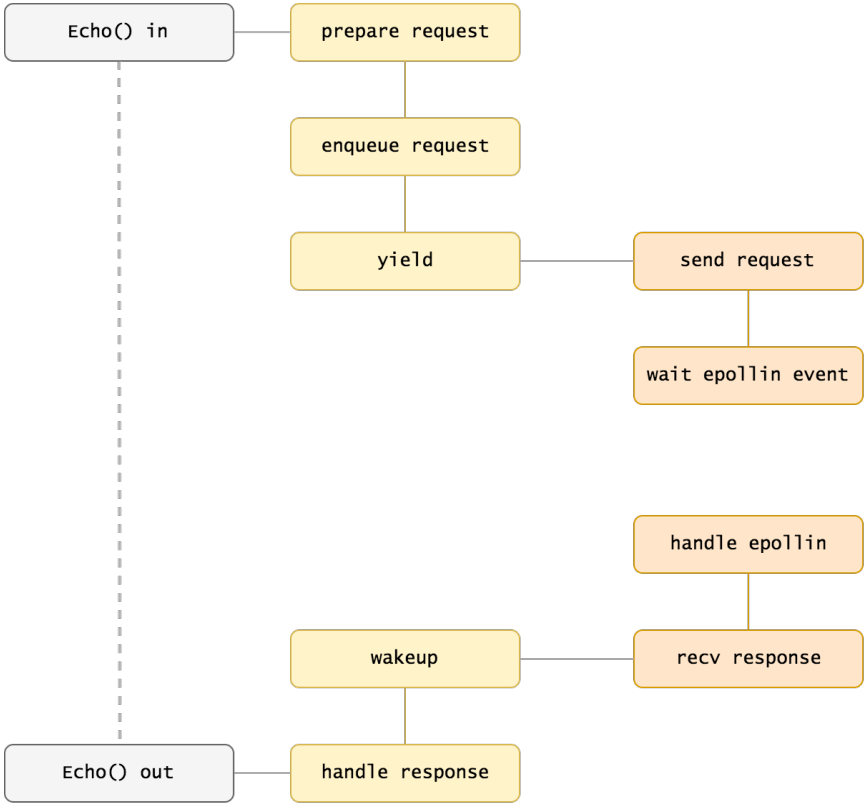
```
void func() {
    Controller   ctrl;
    EchoRequest  request;
    EchoResponse response;

    request.set_msg(RandString());
    service.Echo(&ctrl, &request, &response, nullptr);

    CHECK(!ctrl.Failed());
}
```

- No malloc()/free()
- No explicit yielding
- Asynchronous IO

Behind RPC Example



Coroutine at SmartX

smartx
MAKE IT SIMPLE

- Features
  - Stackfulness and Symmetric
  - Individual stack space: 128KB
  - Protection of stack overflow by mprotect()
  - Stack pool per thread to avoid frequently malloc/free
  - siglongjmp() instead of swapcontext()

- Performance
  - Construct & deconstruct: 131ns
  - Switch in & out: 67ns

Coroutine with Threading

smartx
MAKE IT SIMPLE

- Wrap thread as ThreadContext
  - Thread
  - EventLoop
- Coroutine is schedule between threads
- Each function call is an opportunity of scheduling

Coroutine with Threading

smartx
MAKE IT SIMPLE

- Constructor
  - Schedule to specified thread of thread pool
- Deconstructor
  - Schedule back to original thread

```
void compress() {
    ThreadPoolContextGuard g(thpctx_compression);
    // run in thread pool of compression
}

void write() {
    ThreadContextGuard g(thctx_io);
    // run in thread io
}

void func() {
    compress();
    write();
}
```

Agenda

- Background & challenges
- Architecture of SmartX ZBS
- Experiences with SmartX ZBS
- Roadmap

Roadmap

- Next generation userspace storage manager
  - Kernel bypass: DPDK + SPDK
  - Inline compression
  - Inline deduplication
  - Support faster device
- Data protection for multiple datacenters

Follow SmartX on Zhihu

MAKE IT SIMPLE

We are Hiring ☺

jobs@smartx.com

MAKE IT SIMPLE