

Elasticsearch as a Service on Private Cloud

LINE+ Cloud Service

Clusters

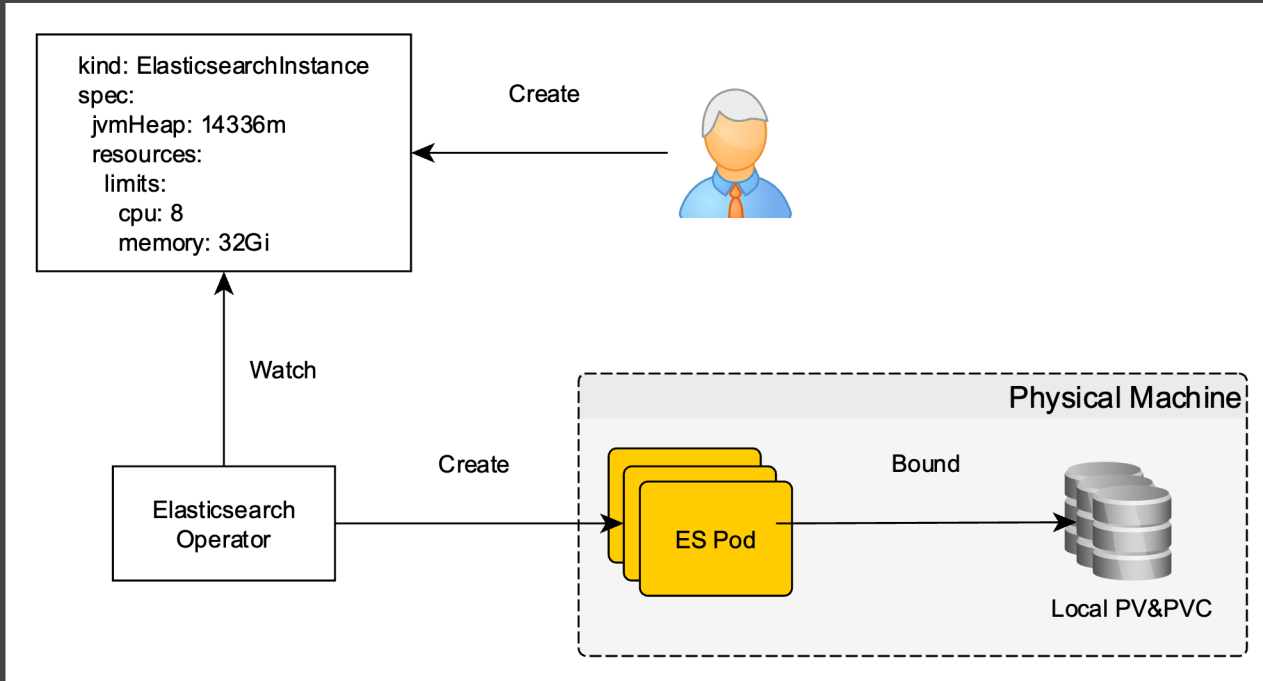
1,500+

Nodes

13,000+

Beginning Architecture

Whole Elasticsearch Clusters on Single Kubernetes Cluster



500 Elasticsearch clusters with
270 Baremetal worker nodes

Declarative approach with
Custom Resources

Troubles...

Don't Put All Your Eggs
in One Basket!

Kubernetes outage resulted in outages of
all Elasticsearch clusters

Traffic burst of one Elasticsearch cluster caused
outage of other services

Troubles...

Don't Put All Your Eggs
in One Basket!

Kubernetes outage resulted in outages of
all Elasticsearch clusters

Traffic burst of one Elasticsearch cluster caused
outage of other services

Requirement 1

Although Kubernetes cluster is down, it should not affect to Elasticsearch clusters

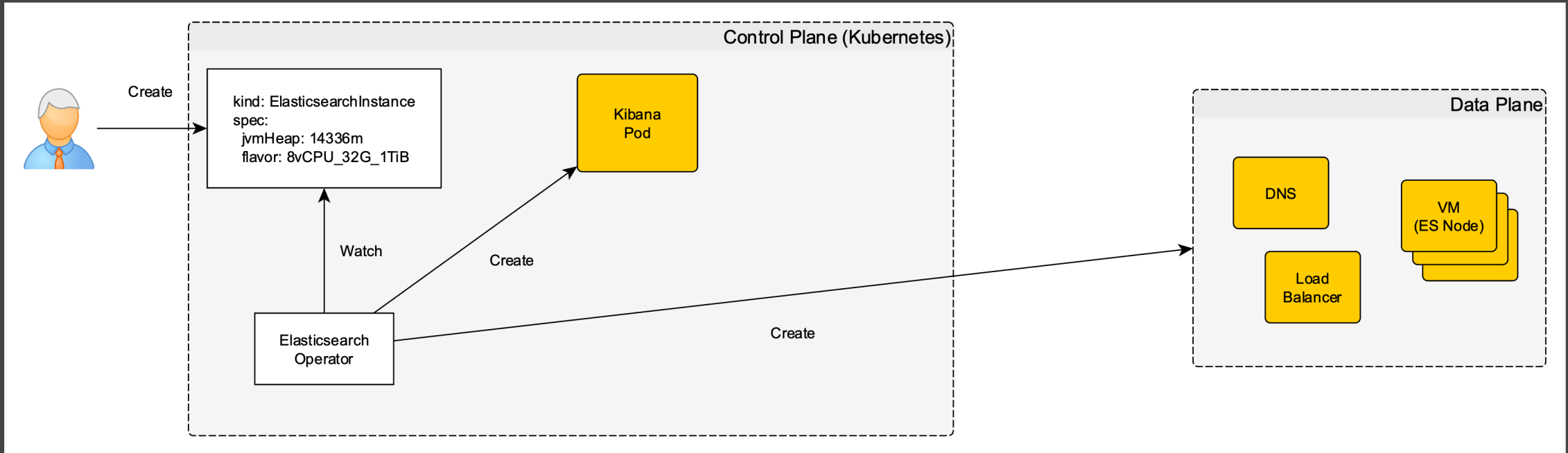
Requirement 1

Although Kubernetes cluster is down, it should not affect to Elasticsearch clusters

Requirement 2

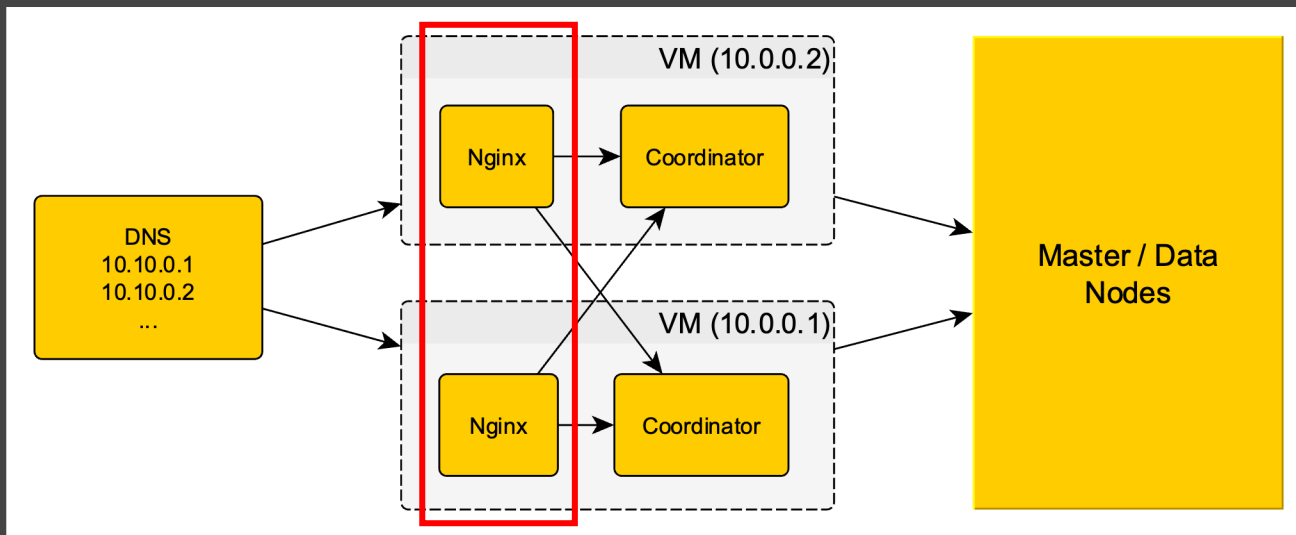
Keep declarative approach of Kubernetes

New Mixed-in Architecture



Control-Plane on Kubernetes, Data-Planes on Virtual Machines

Endpoint Architecture for ES

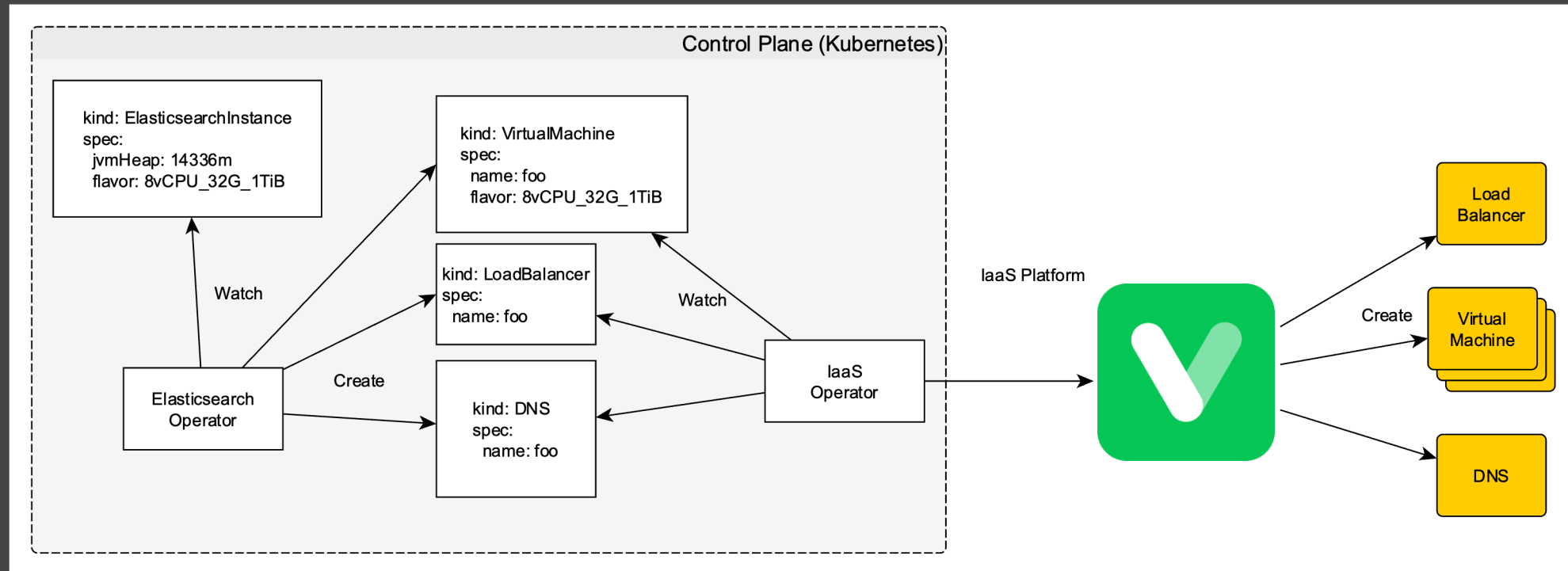


Use Nginx for collecting logs and metrics for each request

DNS round robin + Nginx in clusters for logging

Depend on retrying logic in client

Declarative Approach



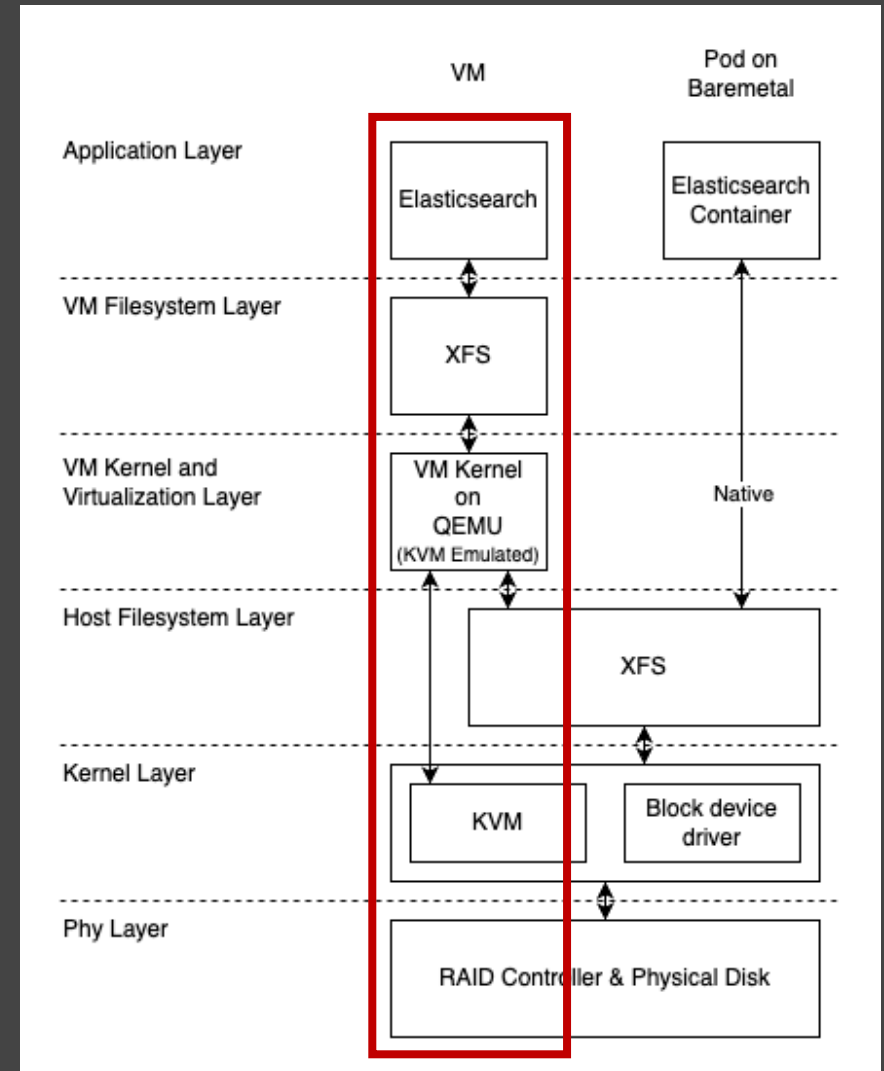
Develop operator to make IaaS resources (VM, DNS, Load Balancers...) declarative objects

**Fulfilled requirements,
another problem arises**

The Limitations

Full-virtualization is really hurt to us

Increasing latency

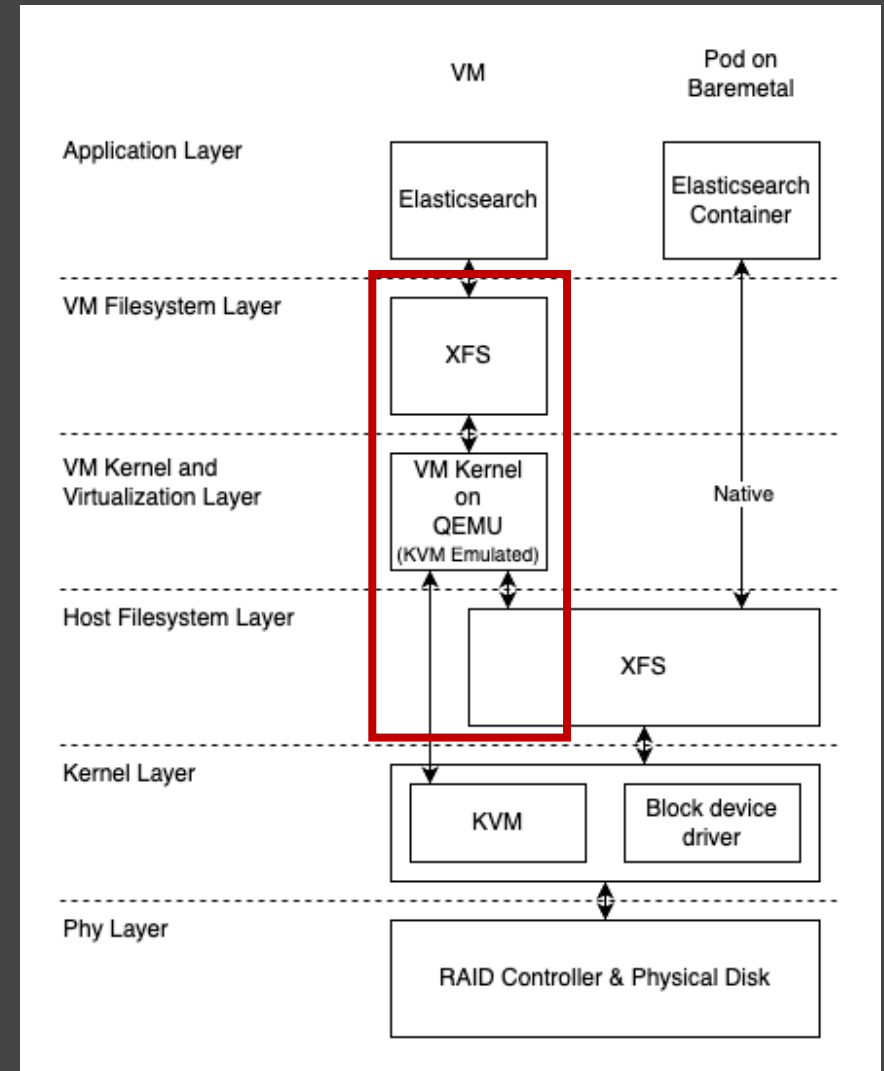


The Limitations

Full-virtualization is really hurt to us

Increasing latency

Decreasing I/O throughput



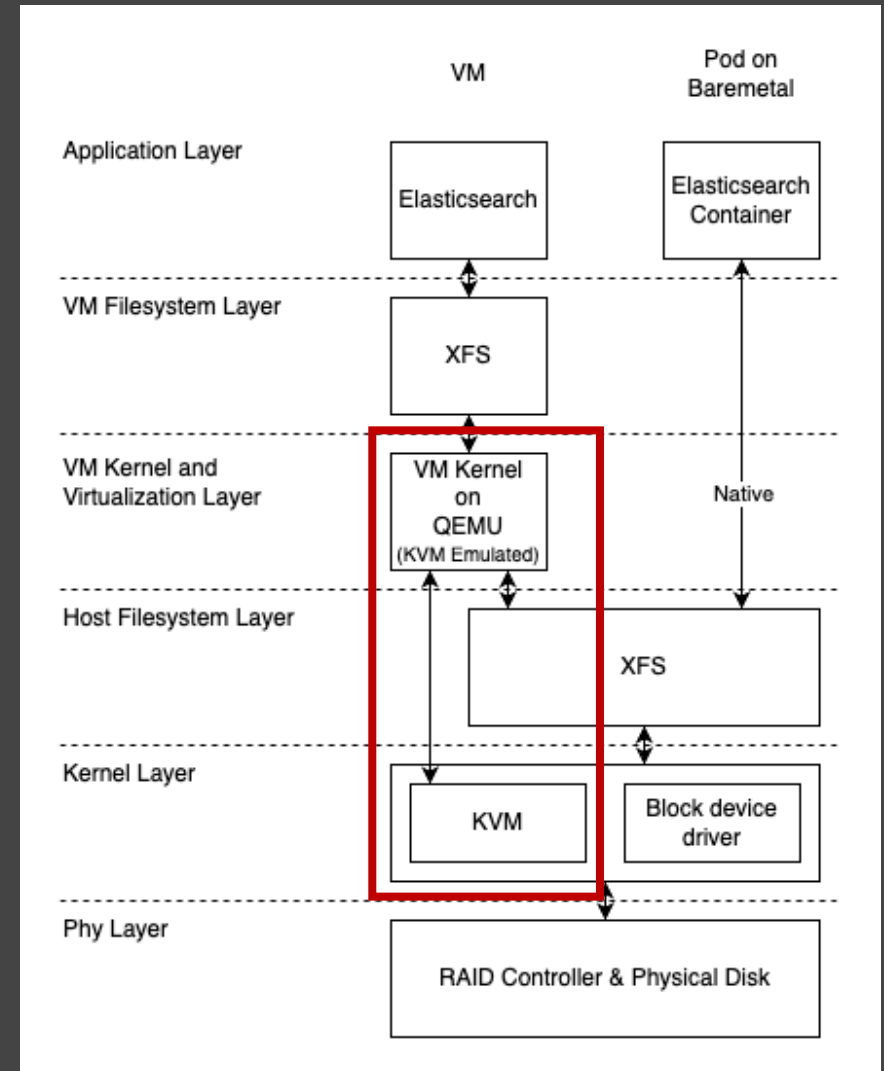
The Limitations

Full-virtualization is really hurt to us

Increasing latency

Decreasing I/O throughput

Bad neighborhood problem



The Limitations

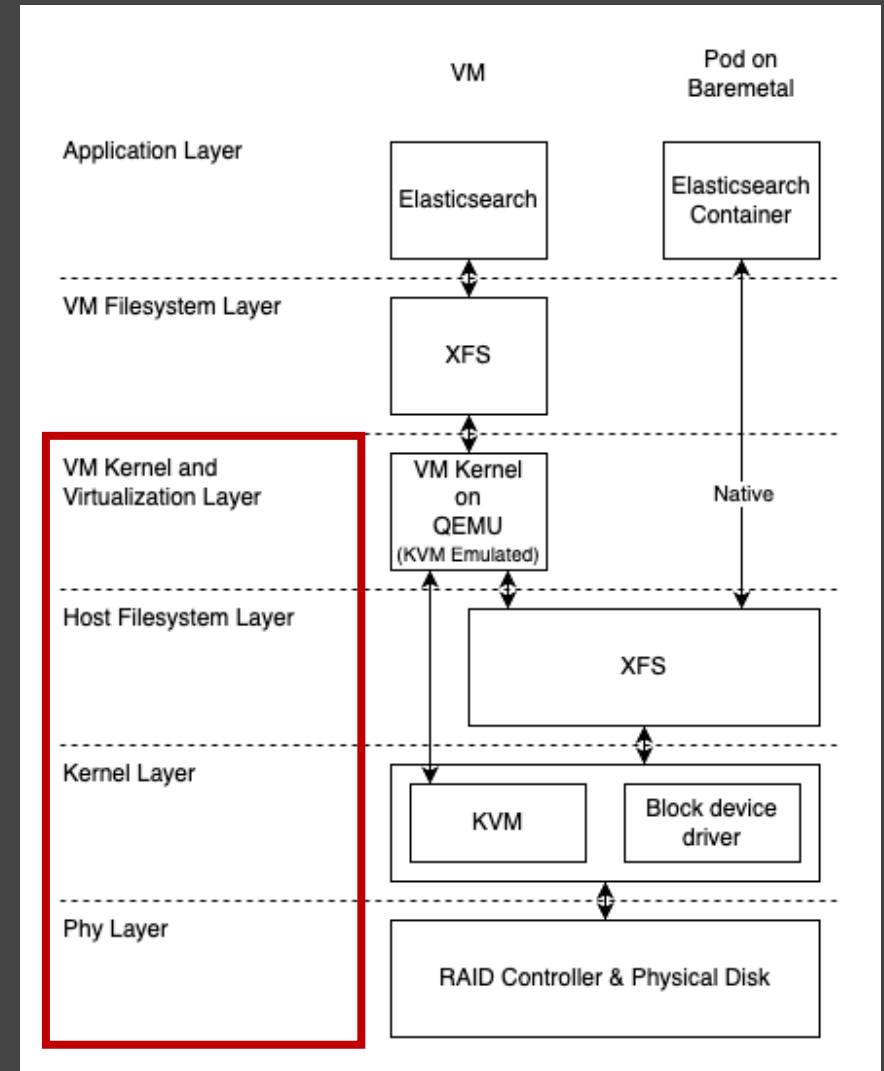
Full-virtualization is really hurt to us

Increasing latency

Decreasing I/O throughput

Bad neighborhood problem

Needs skill set to fine tune



Improve Performance (1)

Tuning Coordinator – Enhance Nginx Service Mesh Throughput

```
CPU [ | 41.4%] CPU 41.4% nice: 0.0% MEM 58.2% active: 1.85G SWAP 0.0% LOAD 12-core
MEM [ | 58.2%] user: 16.9% irq: 0.0% total: 58.8G inactive: 2.15G total: 4.00G 1 min: 5.28
SWAP [ | 0.0%] system: 14.6% iowait: 3.0% used: 34.3G buffers: 2.56M used: 2.01M 5 min: 4.72
idle: 56.1% steal: 0.4% free: 24.6G cached: 3.67G free: 4.00G 15 min: 3.55
```

```
NETWORK Rx/s Tx/s TASKS 247 (536 thr), 1 run, 246 slp, 0 oth sorted automatically by cpu_percent, flat view
_docker0 1.95Gb 1.84Gb
eth0 1.67Gb 1.72Gb
lo 0b 0b
_h05a8620 56Kb 1Kb
_h1be86b0 702Mb 605Mb
_h788ca84 8Kb 528b
_hf045d34 1.93Gb 1.91Gb
```

CPU%	MEM%	VIRT	RES	PID	USER	NI	S	TIME+	IOR/s	IOW/s	Command
176.9	55.3	38.0G	32.5G	4016	elasticse	0	S	5h40:49	0	360K	java -Xshare:auto -Des
60.2	0.0	19.3M	11.8M	36474	101	0	D	10:07.75	52M	53M	nginx: worker process
58.6	0.0	19.3M	11.8M	36473	101	0	S	14:25.43	51M	50M	nginx: worker process
50.7	0.0	18.9M	11.3M	36472	101	0	R	6:32.45	52M	48M	nginx: worker process
47.9	0.0	0	0	64	root	0	S	20:15.50	0	0	ksoftirqd/11
13.5	0.0	17.8M	10.3M	36479	101	0	S	3:56.10	33M	30M	nginx: worker process
13.4	0.0	17.1M	9.60M	36478	101	0	S	1:59.41	24M	27M	nginx: worker process

```
DISK I/O R/s W/s
vda 213M 218M
2022-06-17 13:17:42 No warn.
```

```
CPU [ | 14.7%] CPU 14.7%
MEM [ | 58.2%] user: 10.3%
SWAP [ | 0.1%] system: 2.8%
idle: 84.9%
```

```
NETWORK Rx/s Tx/s TASKS 181 (498 thr), 2 run, 179 slp, 0 oth sorted automatically by cpu_percent, flat view
_docker0 459Mb 389Mb
dummy0 0b 0b
eth0 411Mb 459Mb
lbvip 0b 0b
lo 0b 0b
sit0 0b 0b
tunl0 373Mb 0b
_h05a8620 34Kb 1Kb
_h1be86b0 460Mb 390Mb
_h788ca84 6Kb 352b
_ha72e303 428Mb 428Mb
```

CPU%	MEM%	VIRT	RES	PID	USER	NI	S	TIME+	IOR/s	IOW/s	Command
121.6	55.4	39.4G	32.6G	4016	elasticse	0	S	20h19:00	0	22K	java -Xshare:auto -Des
12.5	0.0	12.2M	4.72M	78645	101	0	S	1h15:43	0	1K	nginx: worker process
9.9	0.0	12.1M	4.58M	78646	101	0	S	41:18.65	0	3K	nginx: worker process
4.8	0.1	1.27G	43.6M	3923	root	0	S	9:55.16	0	0	metricbeat --strict.pe
3.8	0.0	217M	16.0M	41954	root	0	R	18:04.33	0	0	/usr/bin/python /usr/b
3.5	0.0	0	0	64	root	0	S	1h52:55	0	0	ksoftirqd/11
2.2	0.1	1.41G	45.1M	3930	root	0	S	21:04.10	0	19K	filebeat --strict.perm
1.0	0.1	1.44G	43.9M	2288	root	0	S	5:40.56	0	0	/usr/bin/containerd
0.6	0.1	1.58G	75.4M	2303	root	0	S	30:25.66	0	17K	/usr/bin/dockerd -H fd
0.6	0.0	696M	7.74M	3983	root	0	S	3:05.48	0	0	/usr/bin/containerd-sh

```
DISK I/O R/s W/s
vda 0 279K
```

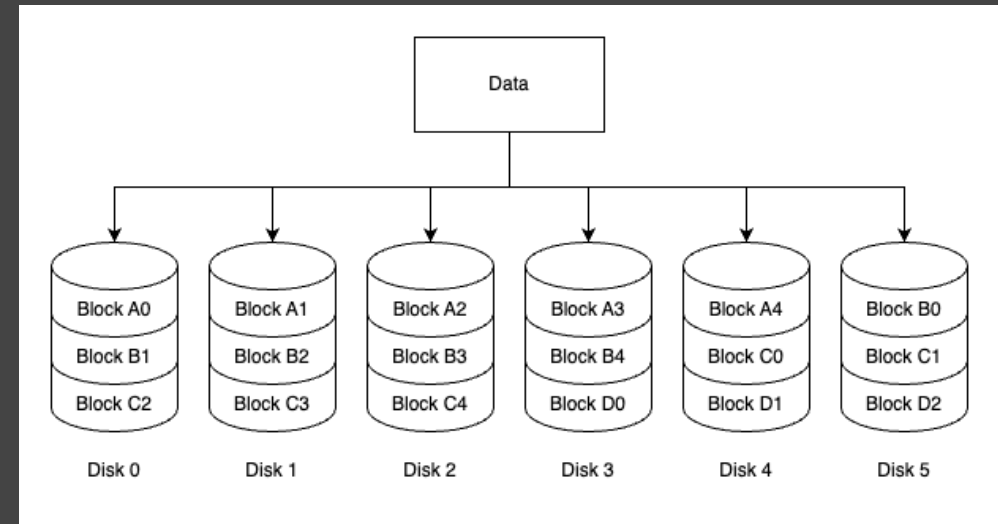
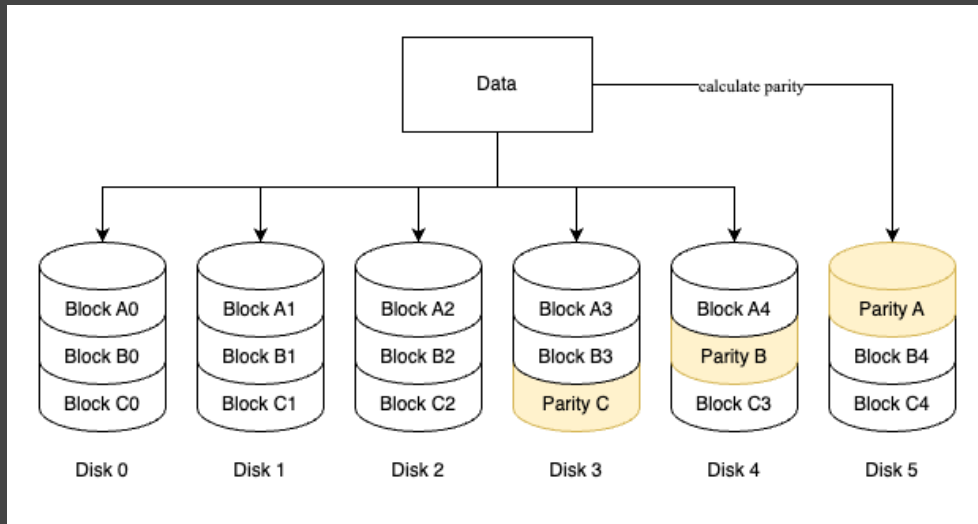
proxy_buffering and proxy_request_buffering

- Larger payloads could be buffered to disk when buffer memory size is smaller than response
- This makes Request-Resp action I/O-intensive
- Performance largely hurts on high RPS with large request and response

```
http {
    ...
    proxy_buffering off;
    proxy_request_buffering off;
    ...
}
```


Improve Performance (2)

RAID 0 – Why we don't care about redundancy











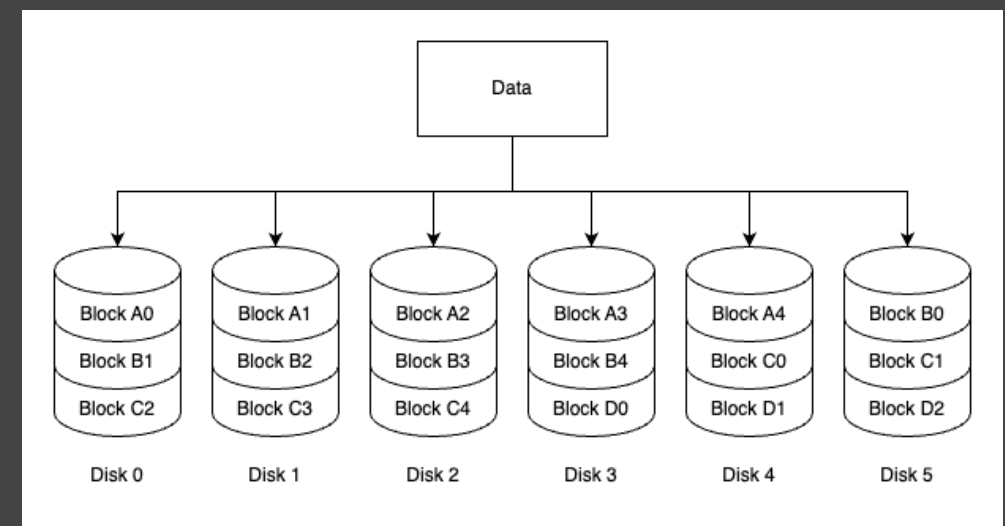
Improve Performance (2)

RAID 0 – Why we don't care about redundancy

Elasticsearch itself has features for high availability

- Elasticsearch has a replica shards
- Lost all VMs disk since RAID 0 has no fault tolerance

shards: 12 * 2 docs: 53,630 size: 102.11MB	shards: 12 * 2 docs: 113,845 size: 145.08MB	shards: 12 * 2 docs: 65,689 size: 104.95MB
		
		
		



Improve Performance (2)







RAID 0 – Why we don't care about redundancy

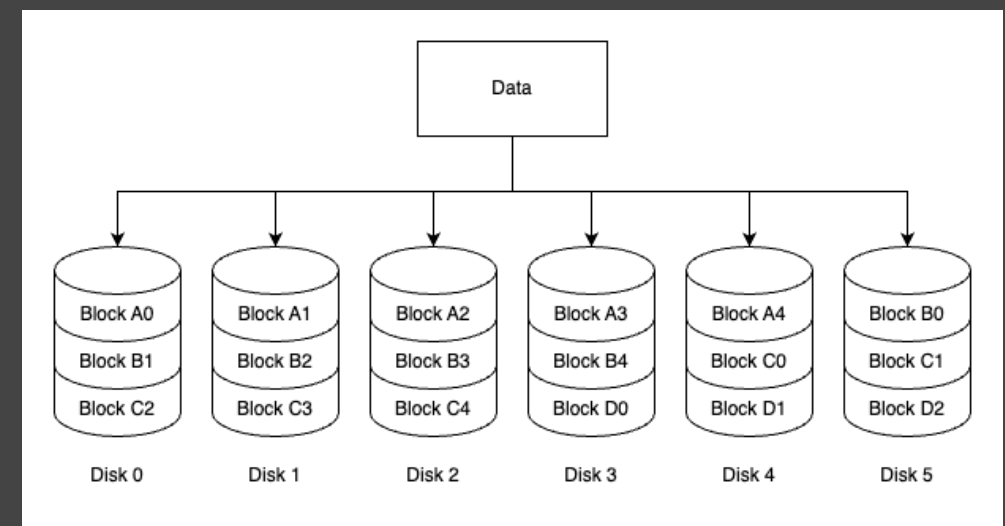
Elasticsearch itself has features for high availability

- Elasticsearch has a replica shards
- Lost all VMs disk since RAID 0 has no fault tolerance

Some clusters need high indexing performance

- Heavy indexing requires high Disk I/O
- Ensure 34,000 IOPS for read and 22,500 IOPS for write in worst case

shards: 12 * 2 docs: 53,630 size: 102.11MB	shards: 12 * 2 docs: 113,845 size: 145.08MB	shards: 12 * 2 docs: 65,689 size: 104.95MB
		
		
		



Improve Performance (2)

RAID 0 – Why we don't care about redundancy









Elasticsearch itself has features for high availability

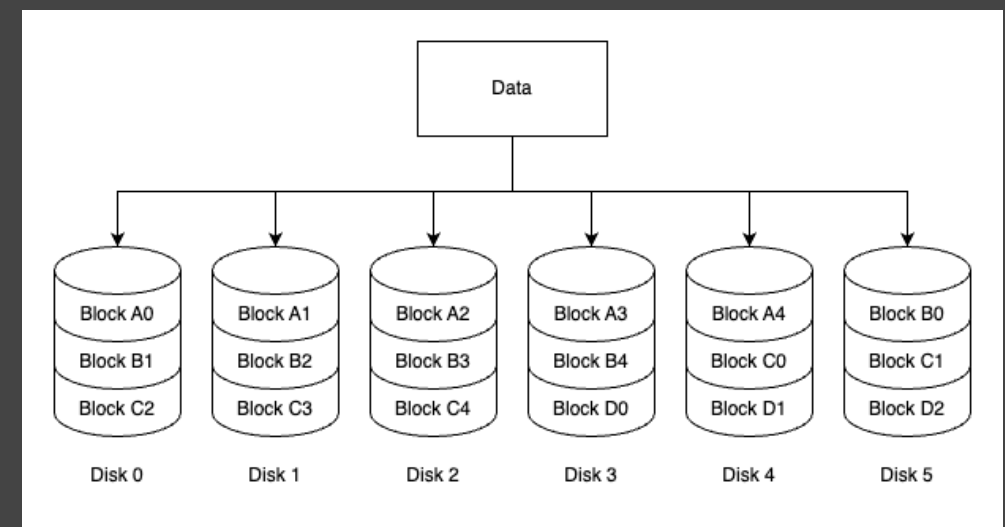
- Elasticsearch has a replica shards
- Lost all VMs disk since RAID 0 has no fault tolerance

Some clusters need high indexing performance

- Heavy indexing requires high Disk I/O
- Ensure 34,000 IOPS for read and 22,500 IOPS for write in worst case

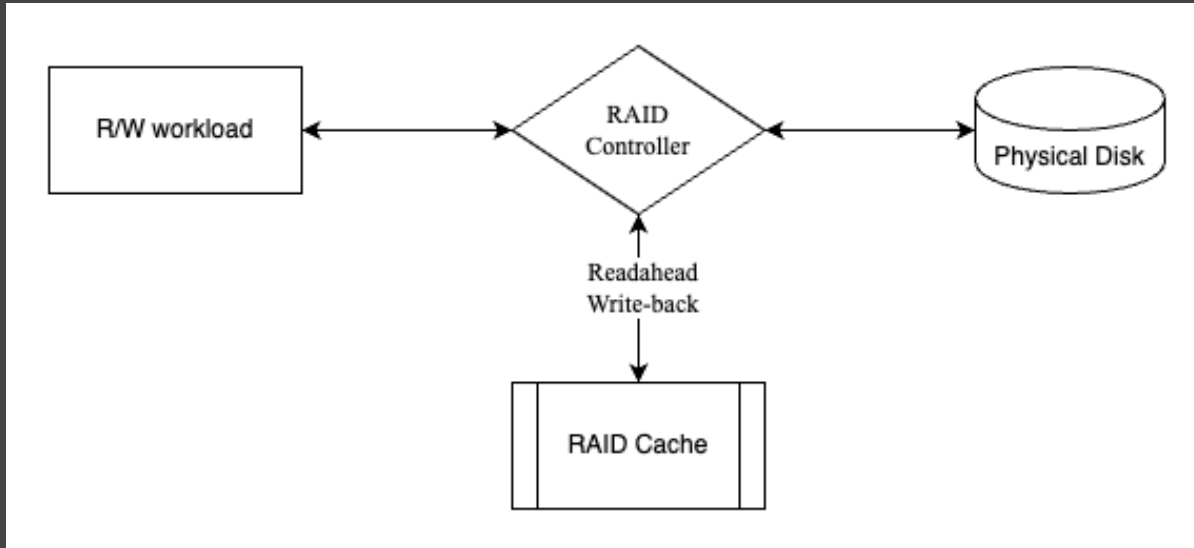
A way to keep redundancy with RAID 0 performance?

shards: 12 * 2 docs: 53,630 size: 102.11MB	shards: 12 * 2 docs: 113,845 size: 145.08MB	shards: 12 * 2 docs: 65,689 size: 104.95MB
		
		
		



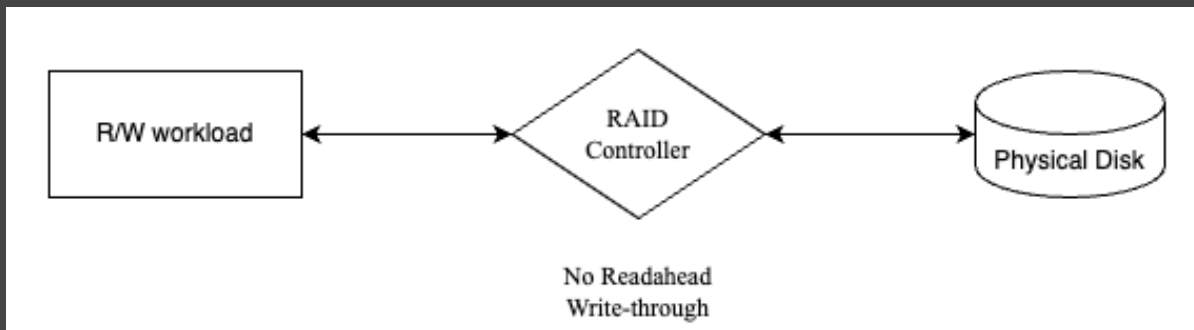
Improve Performance (3)

RAID 0 – Maximizing performance of RAID 0 SSD array



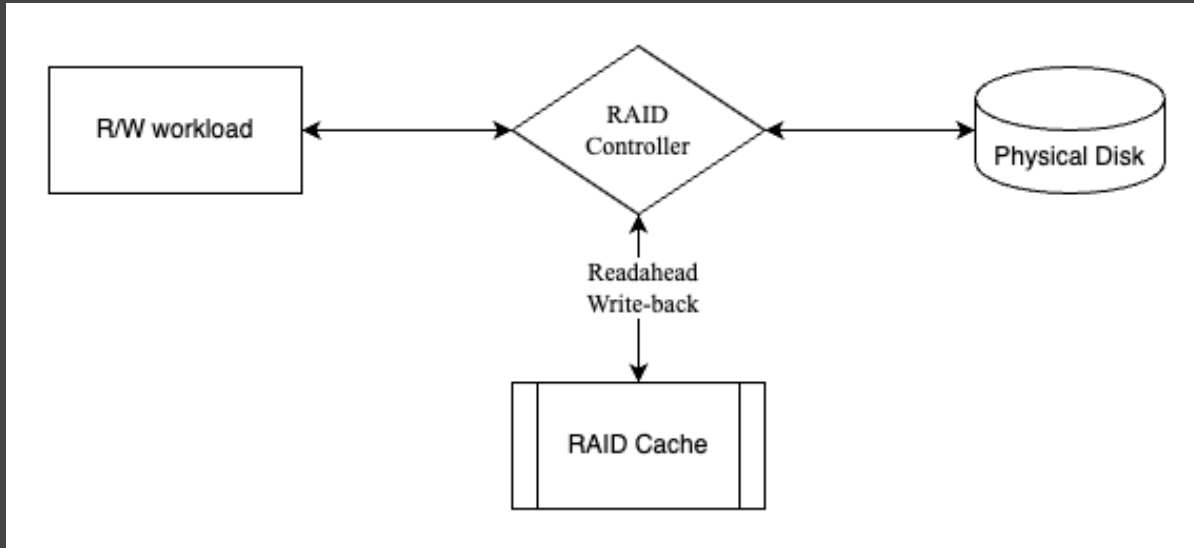
Write-back cache is not always effective

- Dramatically good effect on HDD, but not on SSD



Improve Performance (3)

RAID 0 – Maximizing performance of RAID 0 SSD array

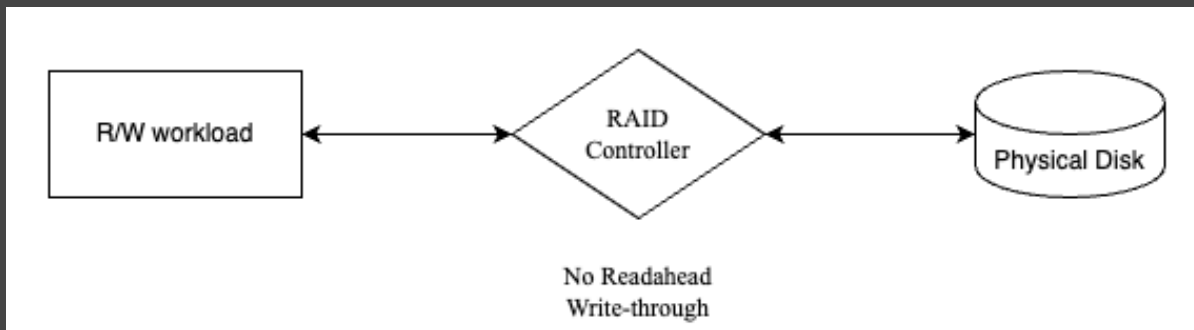


Write-back cache is not always effective

- Dramatically good effect on HDD, but not on SSD

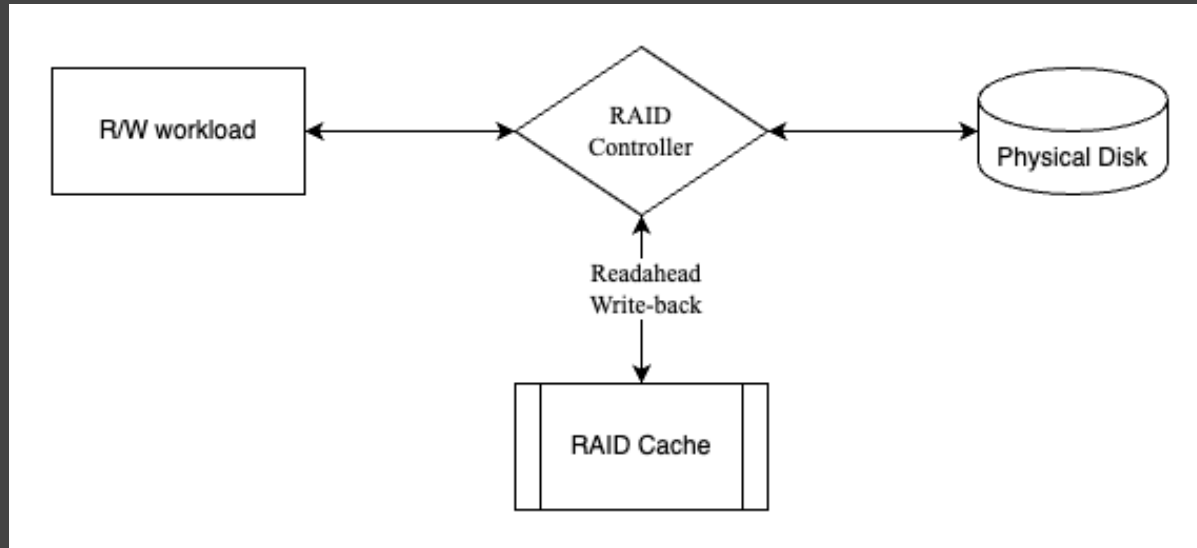
Readahead is not required to SSD

- Locality of data
- Most effective on HDD too



Improve Performance (3)

RAID 0 – Maximizing performance of RAID 0 SSD array



Write-back cache is not always effective

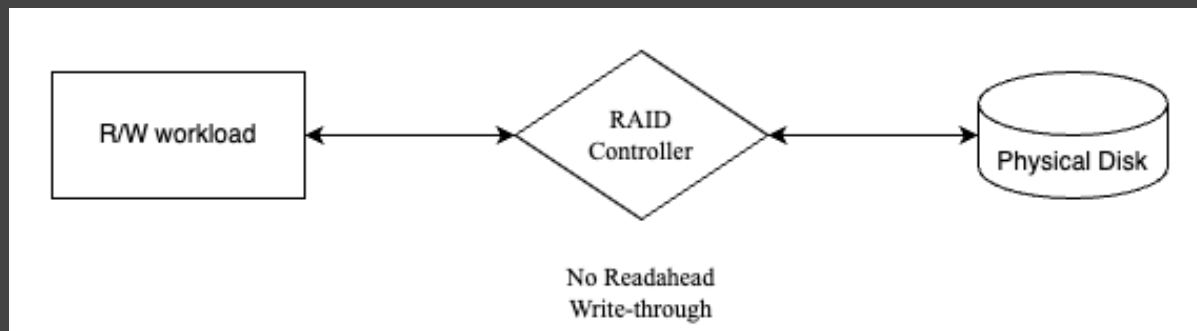
- Dramatically good effect on HDD, but not on SSD

Readahead is not required to SSD

- Locality of data
- Most effective on HDD too

Enabling SSD Array Optimization

- Supported by RAID controller



Results

Random Read

128,000 IOPS → 307,000 IOPS

 **240% Improved**

Random Write

39,600 IOPS → 204,000 IOPS

 **515% Improved**

Improved Disk I/O Performance

*Higher is better
fio 3.7 - 4K randread and randwrite

Results



Indexing

Total Time

5h 32m → 2h 41min



51% Reduced



Searching

Latency

45.65ms → 19.17ms

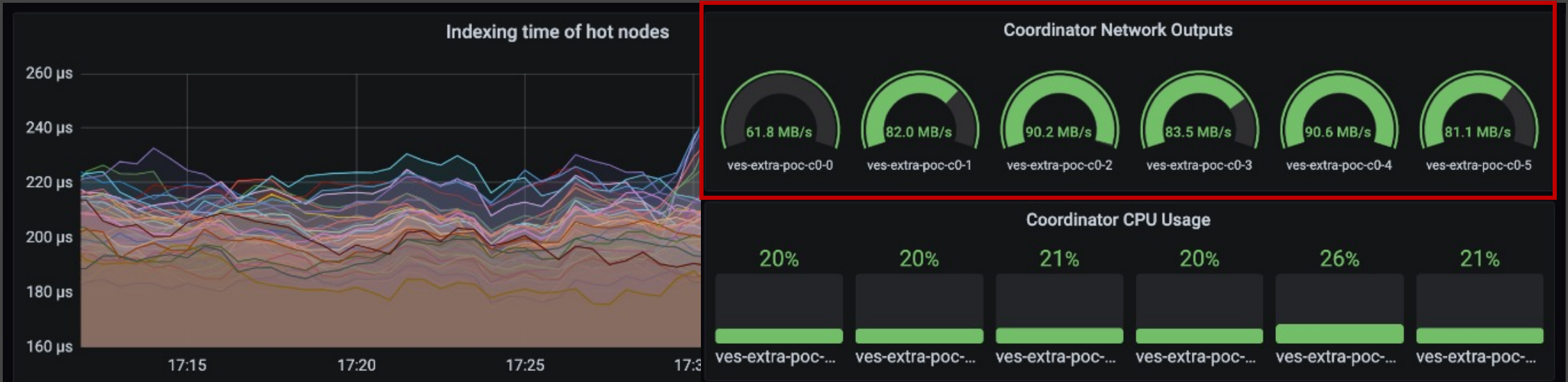


58% Reduced

Improved esrally performance

*Lower is better; Compared with newly designed node type
esrally benchmark set - http_logs and pmc

Results



Metric of proof-of-concept cluster after performance tuning

FUTURE PLANS

OpenSearch

License problems for higher version

Support both Elasticsearch and OpenSearch

Migrate Elasticsearch to OpenSearch without downtime

Performance

Bring the largest legacy cluster in LINE to Private Cloud

Support the multiple container for extreme workloads

Support NVMe-oF to search-intensive clusters

Auto-healing

Replace abnormal nodes automatically

Need different strategy by node role

Node Pool Architecture

- Tolerance for outages of IaaS control plane
- Support baremetal to provide native performance

High Availability

Support multiple availability zone for IaaS resources

Support fault tolerance against datacenter failure

- Cross-Cluster Replication
- Place role-set to every availability zone (DC)



We are hiring



No need for Elasticsearch / OpenSearch Experience

