Accelerating Distributed Deep Learning Inference on multi-GPU with Hadoop-Spark

Hunmin Yang, Se-Yoon Oh, Ki-Jung Ryu

March 2019

GTC Silicon Valley

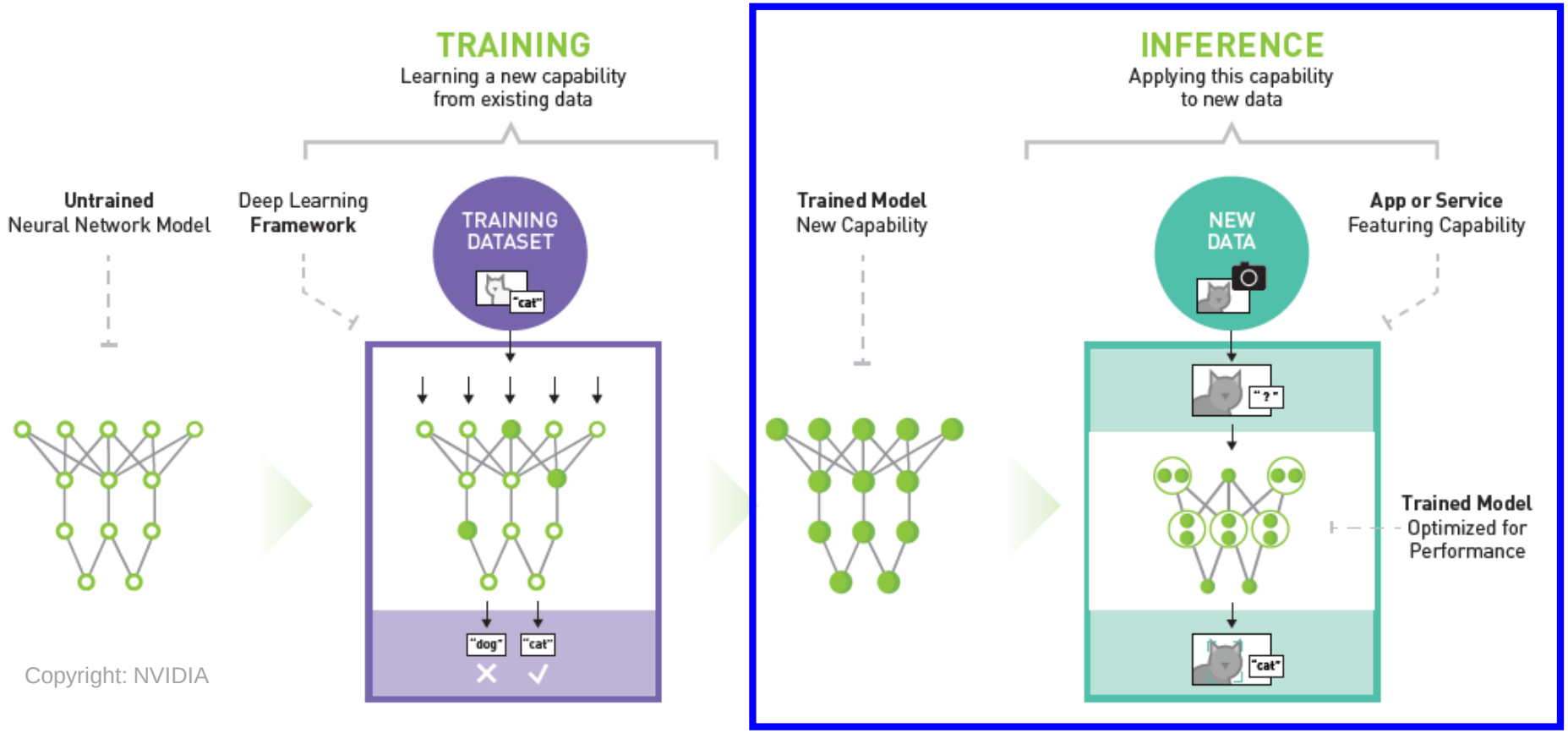
Agency for Defense Development

Agenda

- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions

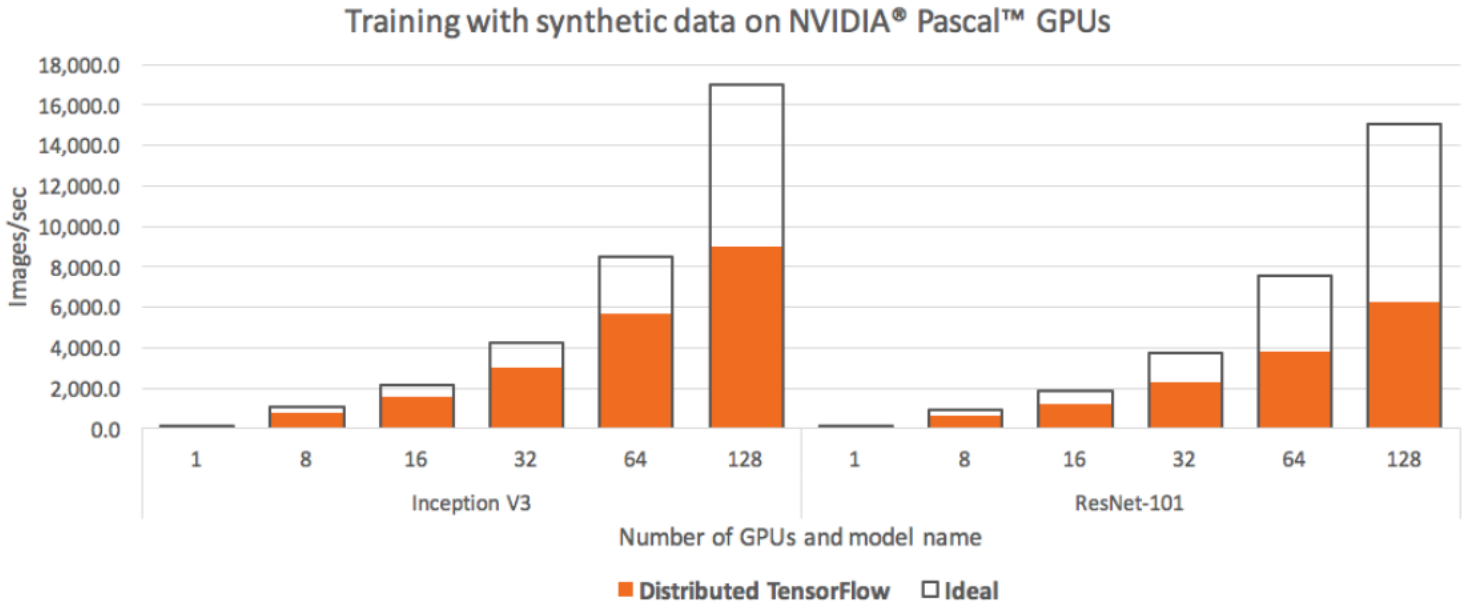
Introduction

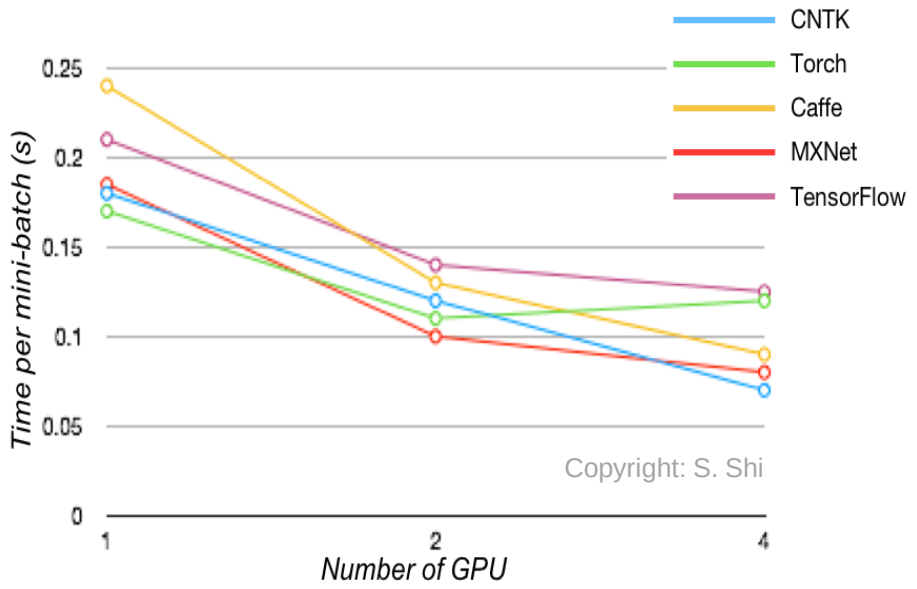
- AI Training & Inference
- Which one is more important for real-world applications?



Introduction

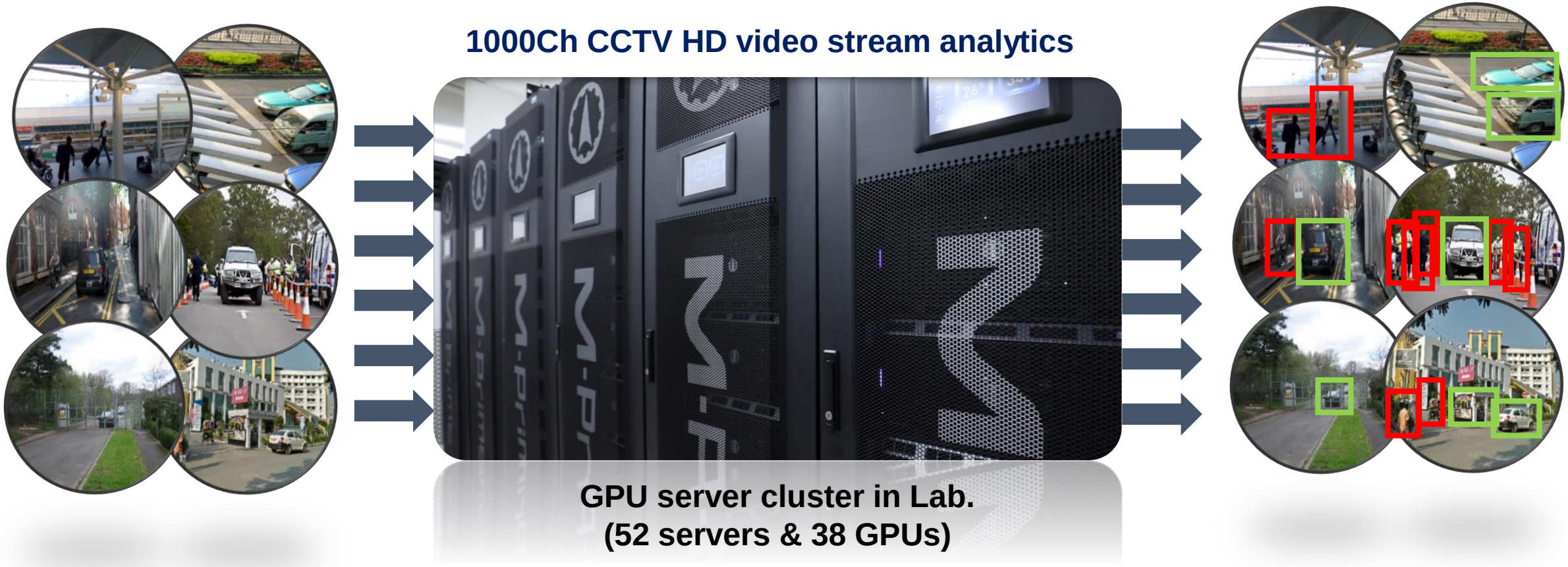
- Especially for industrial applications, large-scale AI inference is needed.
- Then, large-scale AI inference is easy?
  - Not really.
  - Linear scalability is not guaranteed for multiple GPUs and Servers.
  - Furthermore, most works are rather focused on AI training.





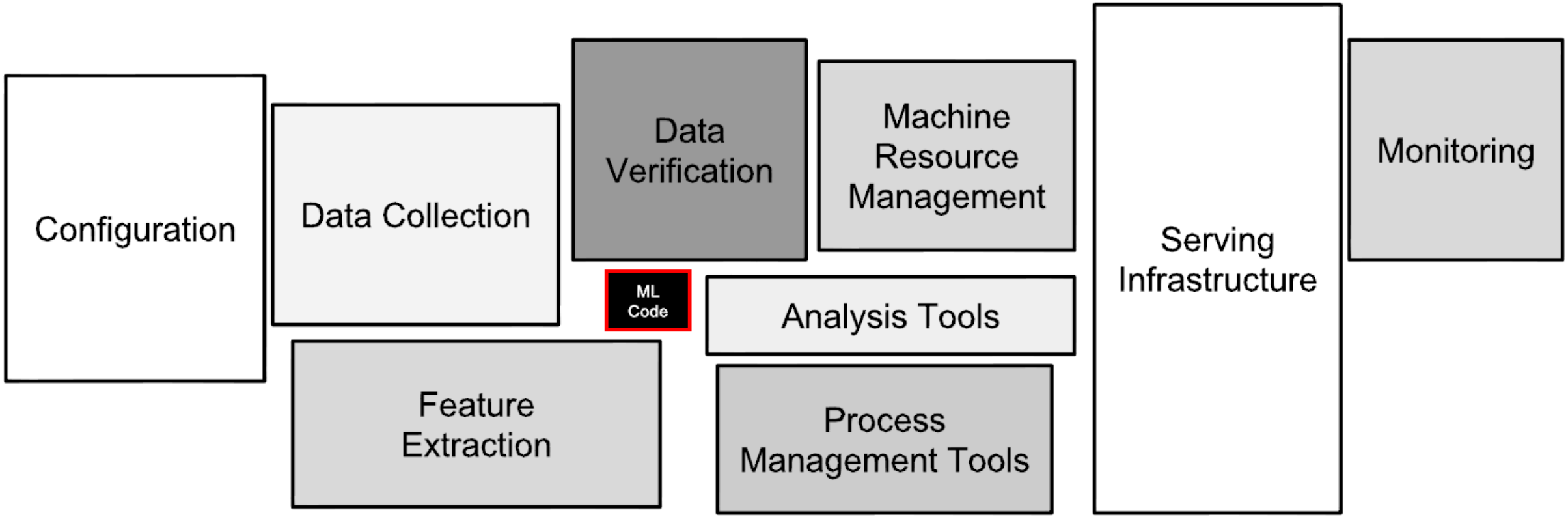
Motivation

- Need for developing a large-scale production-level deep learning inference platform for intelligent video analytics



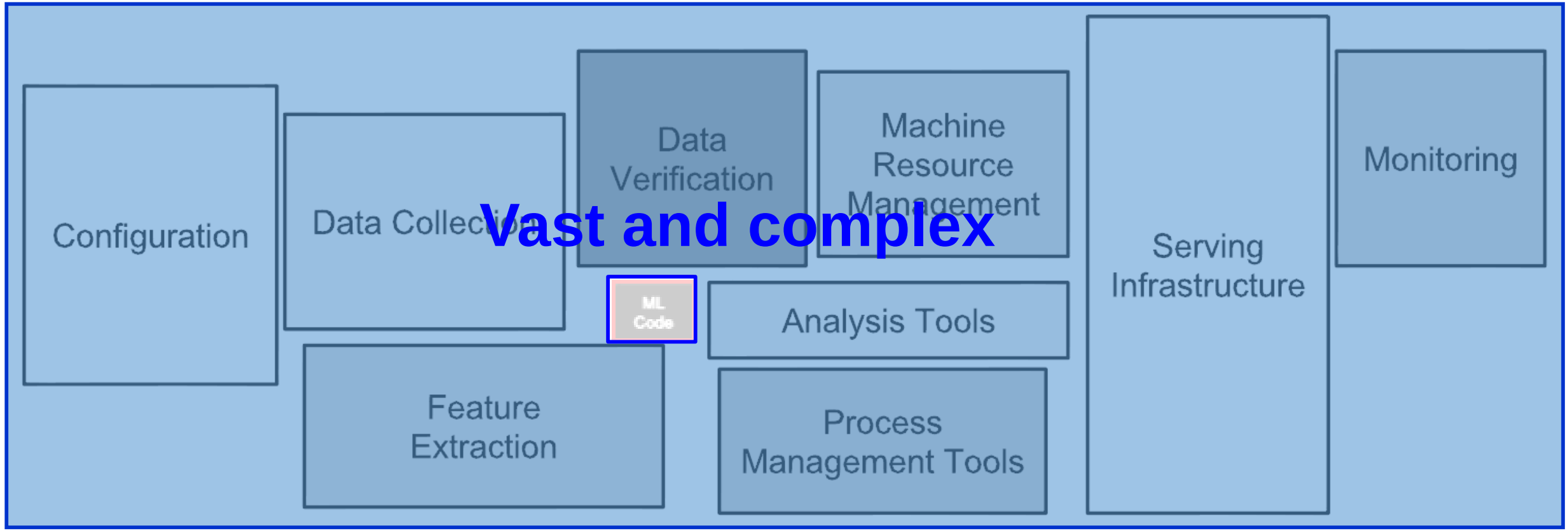
Trend #1

- Real-world AI systems are complex big data analytics pipelines
  - Only a small fraction of real-world ML systems is composed of the ML code.
  - The required surrounding infrastructure is vast and complex.



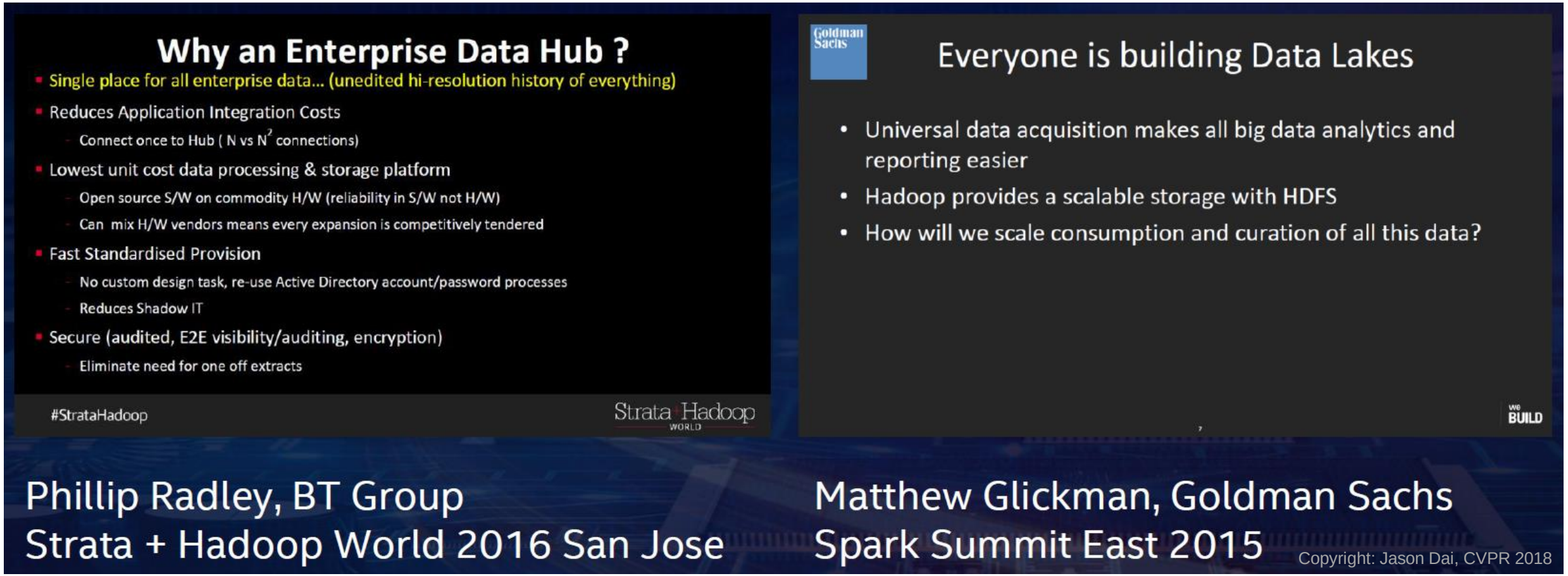
Trend #1

- Real-world AI systems are complex big data analytics pipelines
  - Only a small fraction of real-world ML systems is composed of the ML code.
  - The required surrounding infrastructure is vast and complex.



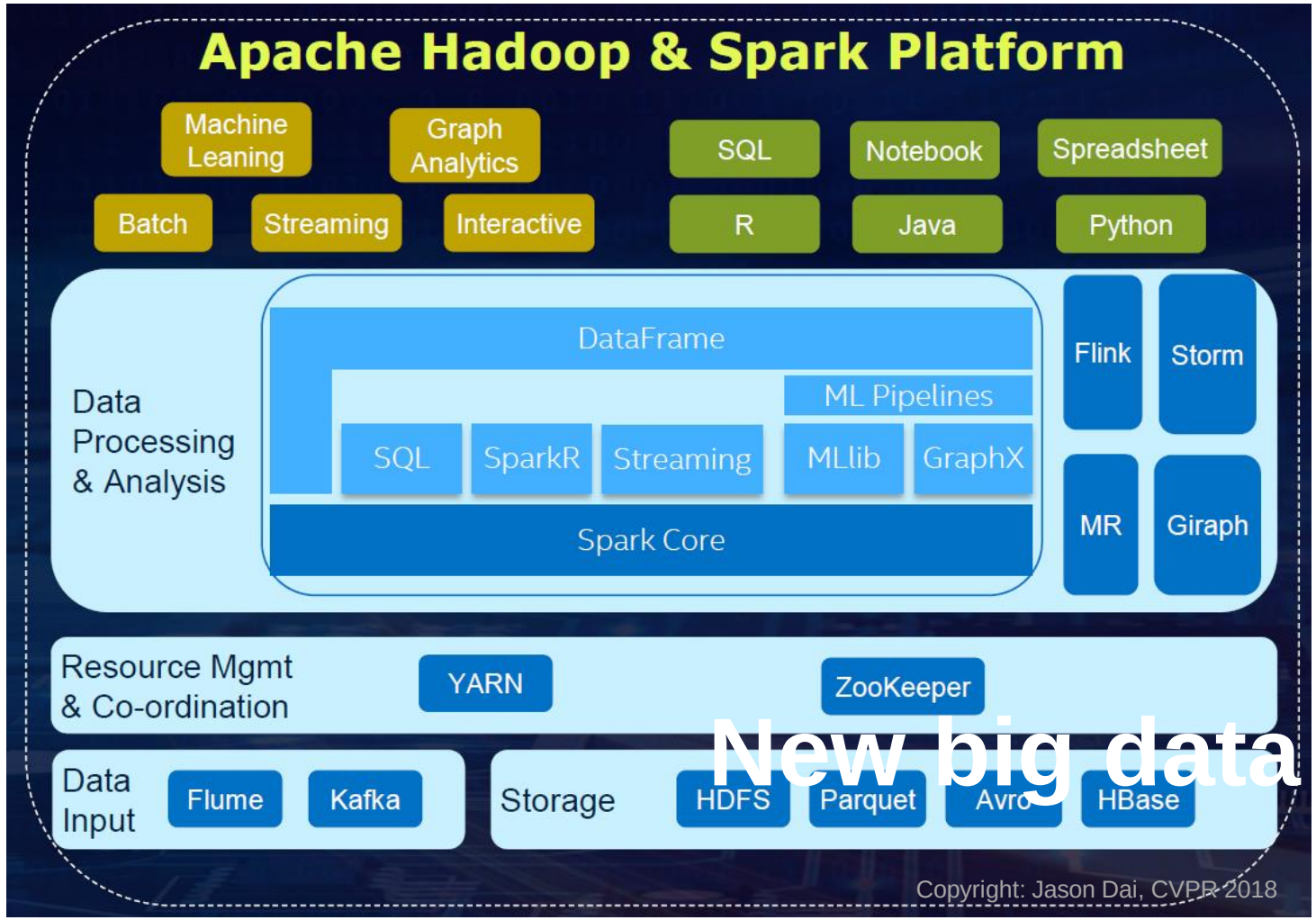
Trend #2

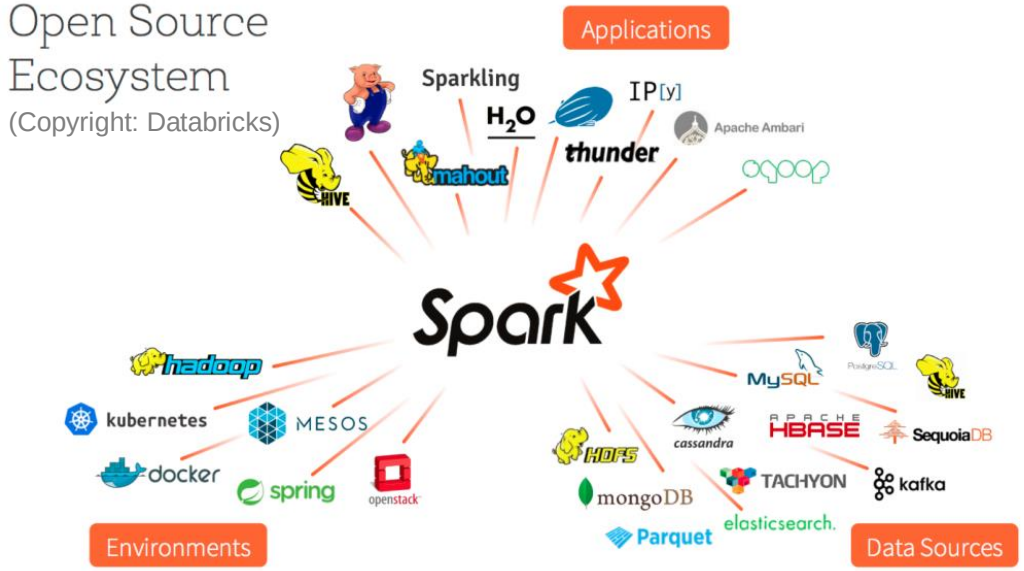
- Hadoop becoming the center of data storage (DB)



Trend #3

- Hadoop and Spark becoming the center of data processing (analytics)



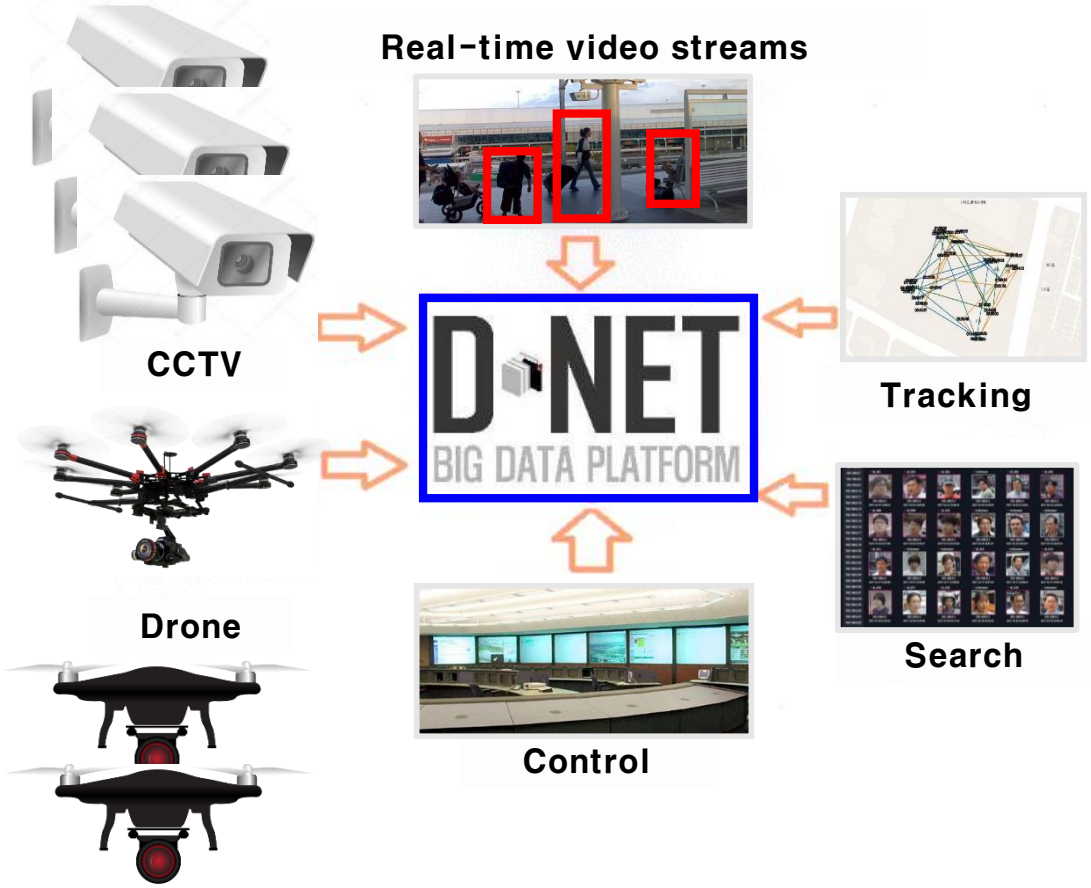


Spark contributors(up to ‘19.2)

- **Over 300 companies**
- **Over 1200 developers**
- **Over 25 organizations**

Research objective

- Develop a fast, scalable, and optimal GPU-accelerated distributed deep learning inference platform with Hadoop-Spark



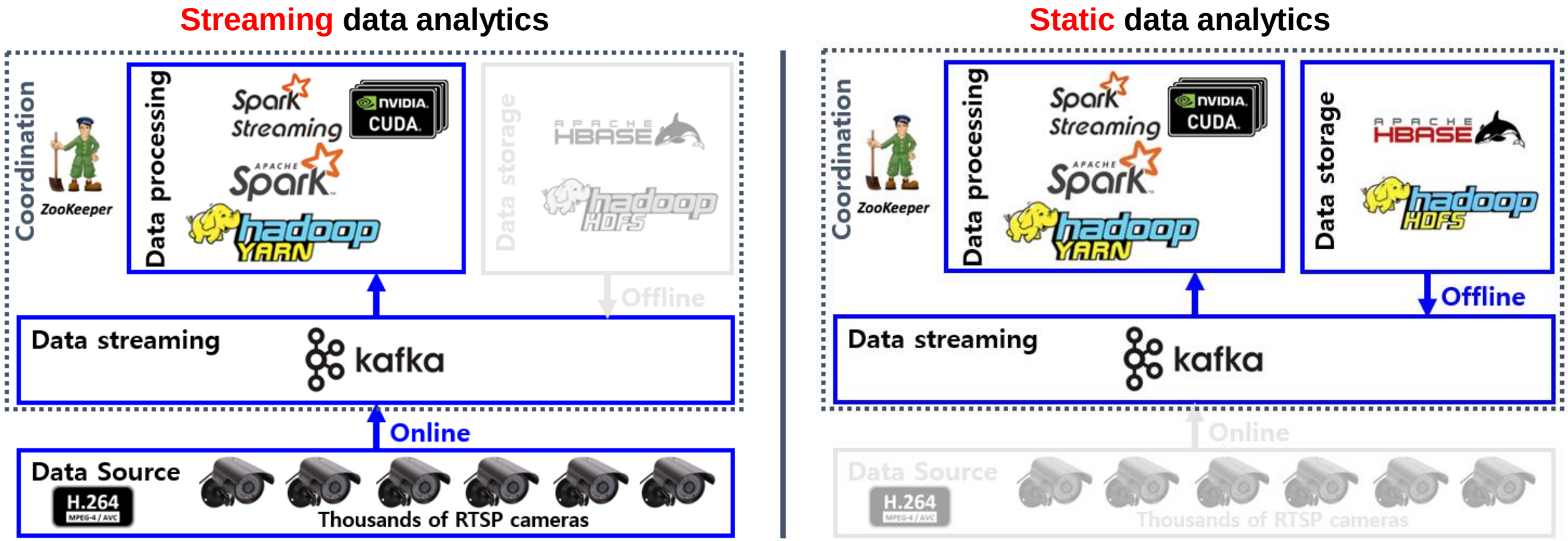
AI application deployment at scale

Outline

- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions

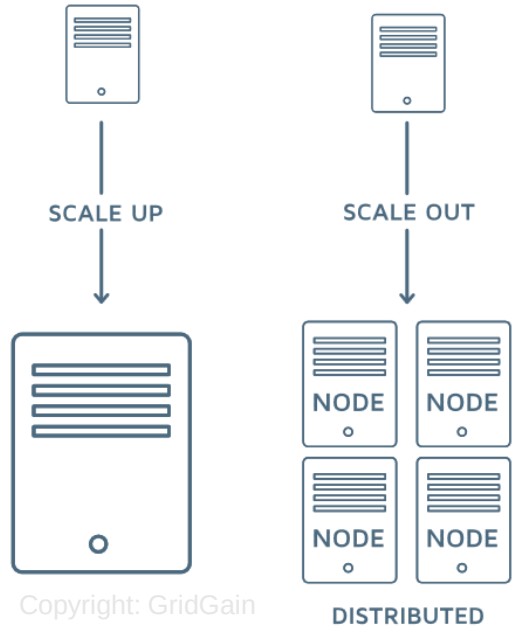
Part 1 - Overview

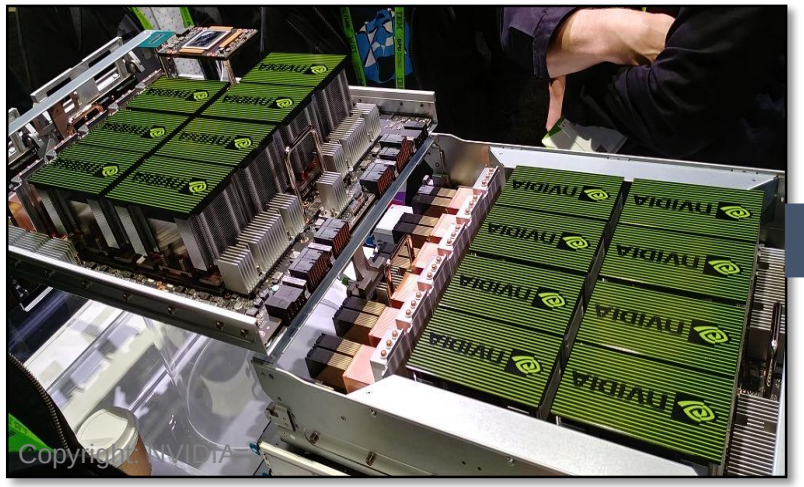
GPU-accelerated DL Inference with Hadoop-Spark



Challenge: Fast & Scalable DL Inference

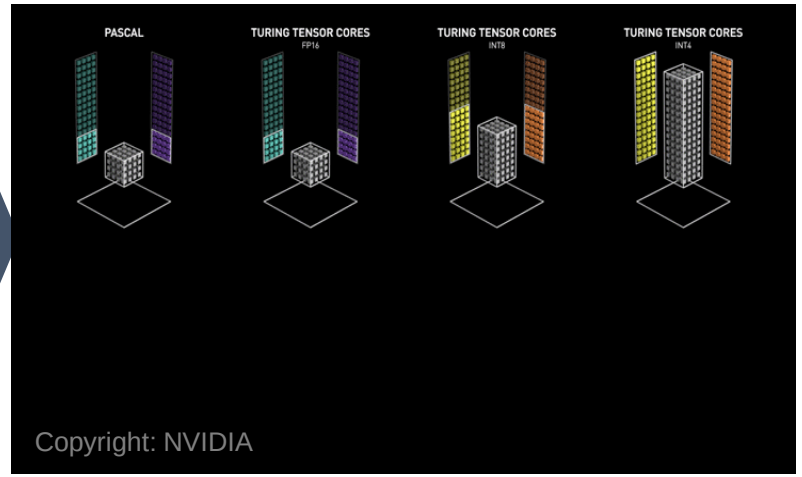
- Why do we need ‘fast’ DL inference?
  - Low latency for high-quality service
  - Especially critical for defense industry
- Why do we need ‘scalable’ DL inference?
  - Good scalability means fast computing
  - ‘Scale up’ - one server with higher spec.
  - ‘Scale out’ - many servers with clustering





Multi-GPU

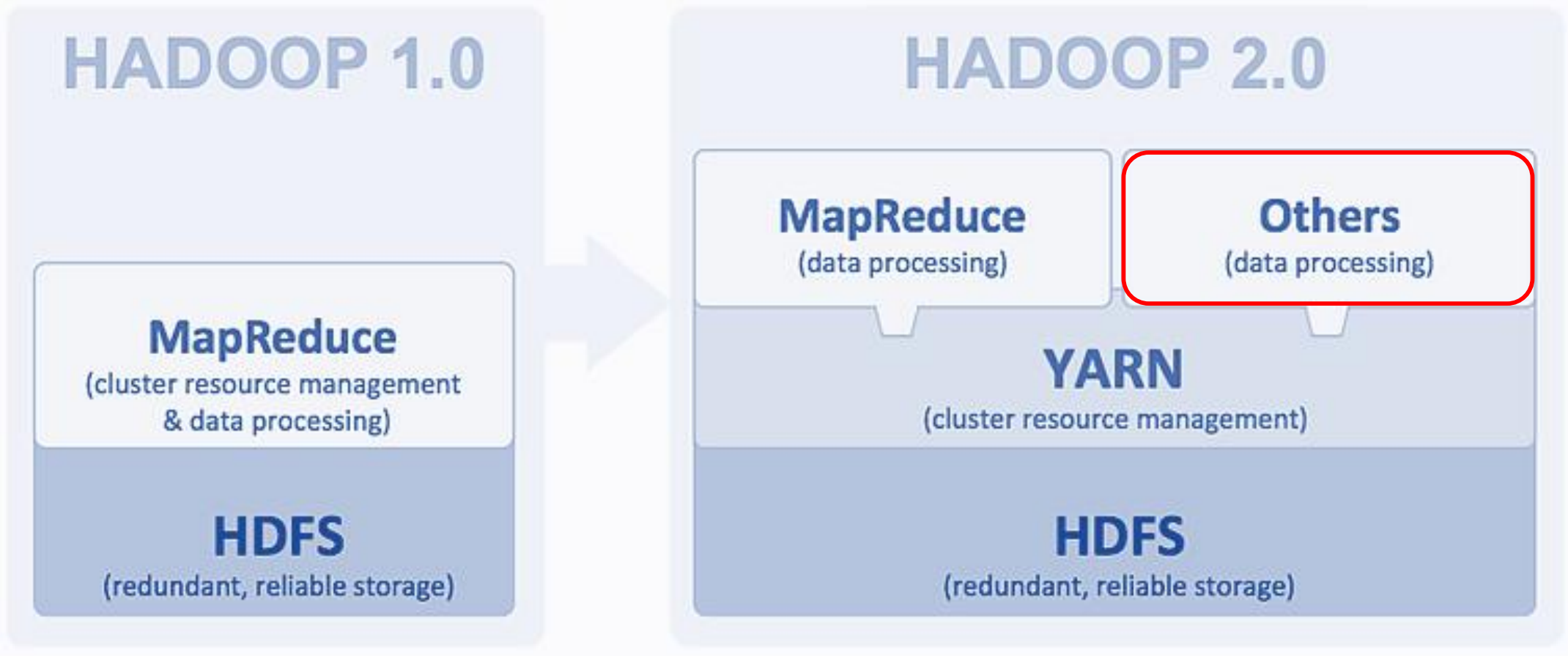
Multi-node



Fast service

Challenge: Fast & Scalable DL Inference

- SOTA distributed computing: Apache Hadoop

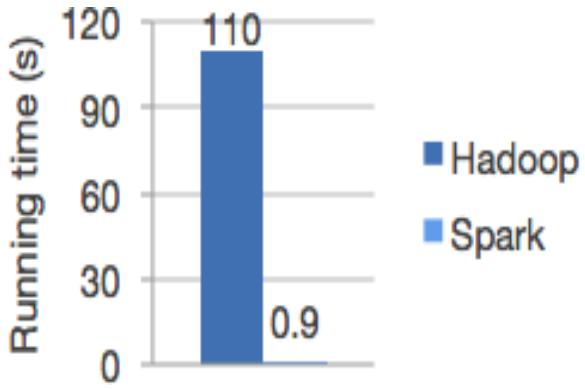


Single-use platform
- Batch processing

General-purpose platform
- Batch processing
- Streaming / Online / Interactive processing

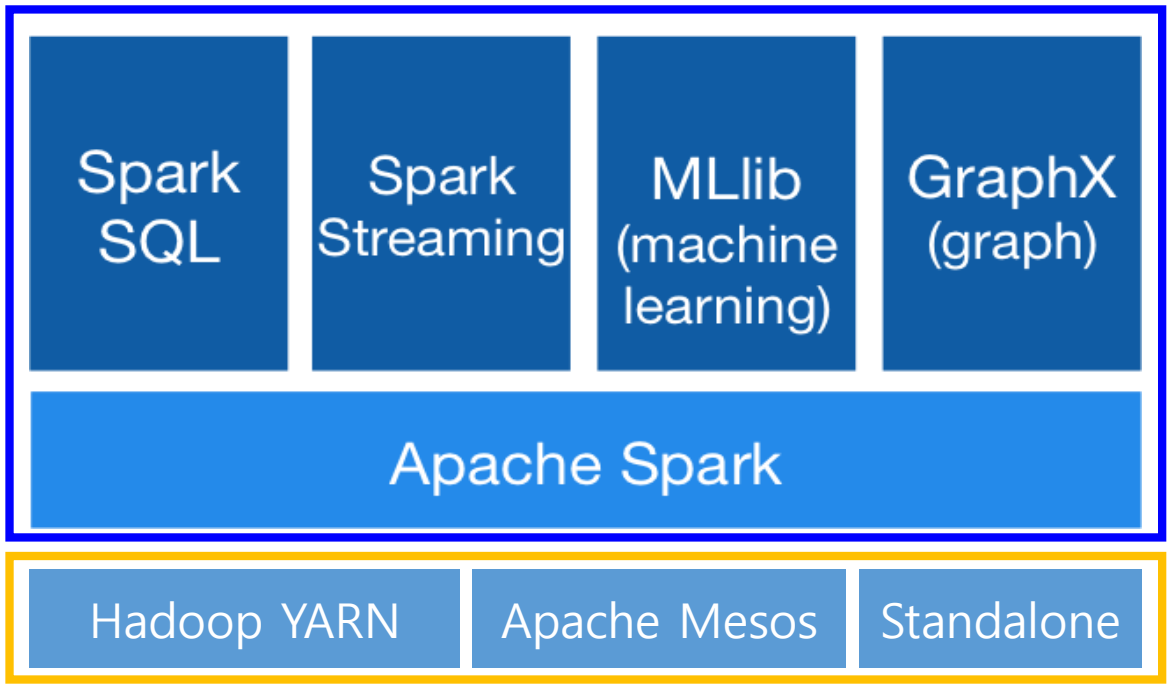
Challenge: Fast & Scalable DL Inference

- SOTA distributed computing: Apache Spark
  - Fast & General-purpose cluster computing engine
  - With Hadoop YARN, Apache Mesos, Standalone, etc.
  - High-level APIs in Java, Scala, Python, R, and SQL



SOTA
- DAG scheduler
- query optimizer
- physical execution engine

Fast

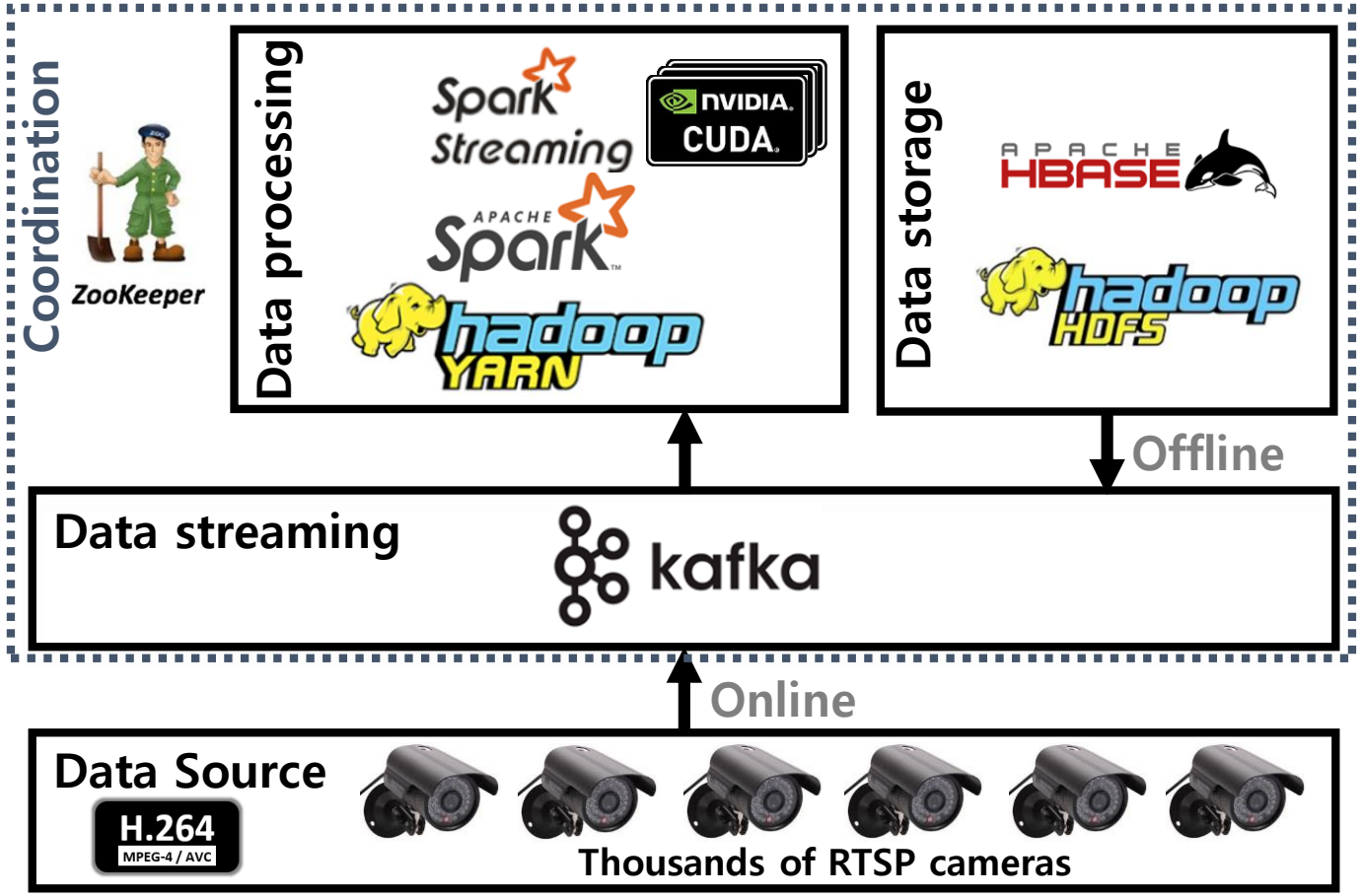


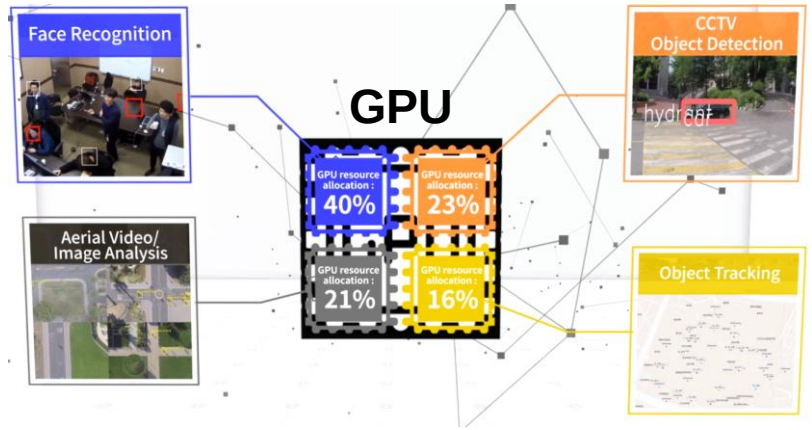
General-purpose

Runs-everywhere

Solution: NVIDIA GPU & Hadoop-Spark

- Our solution
  - Hadoop-Spark: efficient distributed computing
  - NVIDIA GPUs: accelerating DL inference





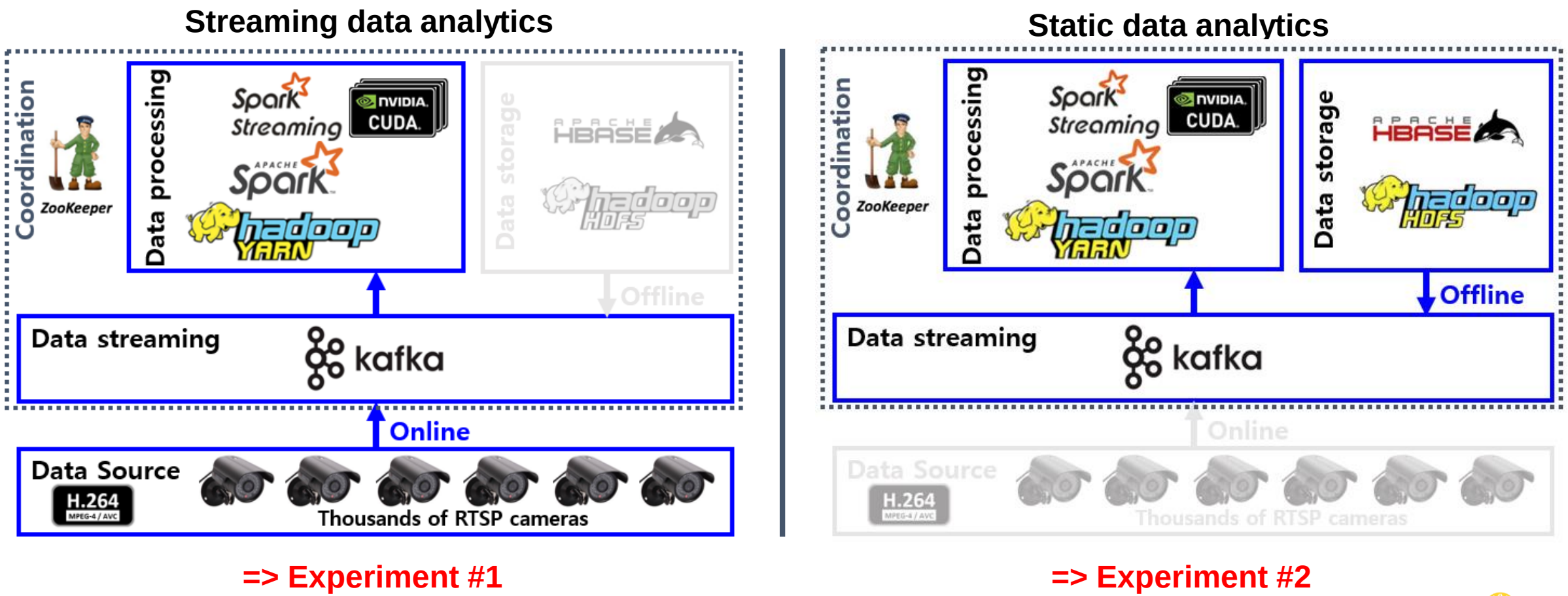
Main benefits

- ❖ Efficient GPU resource allocation among cluster servers with Hadoop **YARN**
- ❖ Better GPU occupancy by multi-processing in each GPU with **Spark**
- ❖ Faster data access with **HDFS** and **HBase**

Solution: NVIDIA GPU & Hadoop-Spark

- Our solution

GPU-accelerated DL Inference with Hadoop-Spark

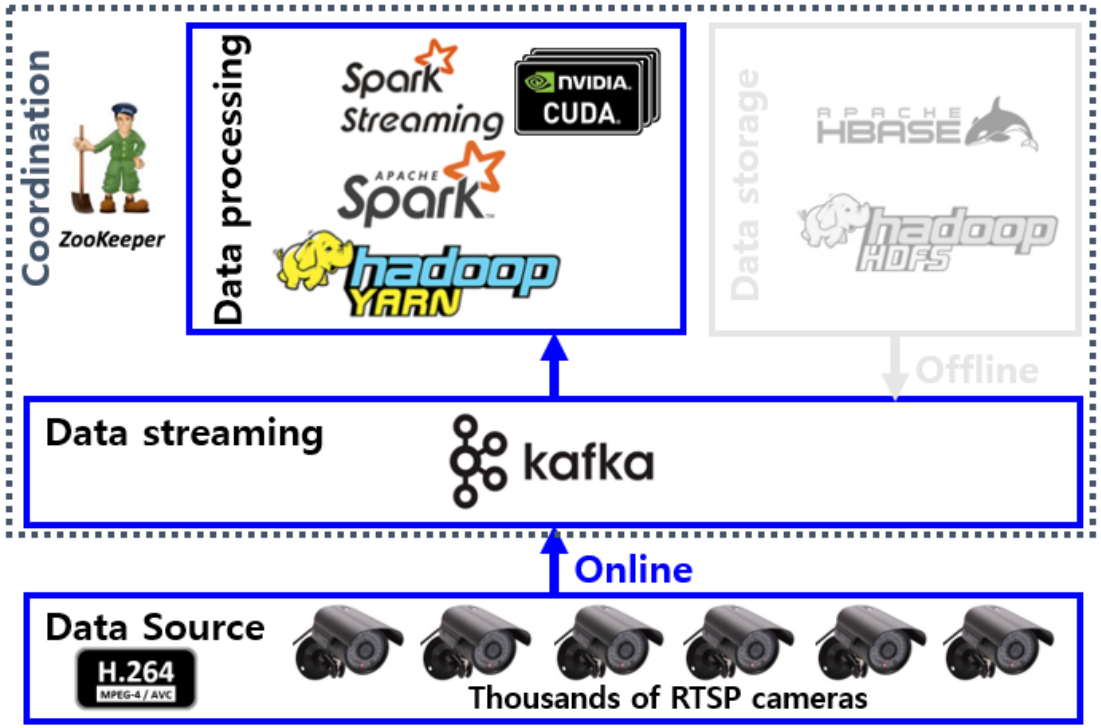


=> Experiment #1

=> Experiment #2

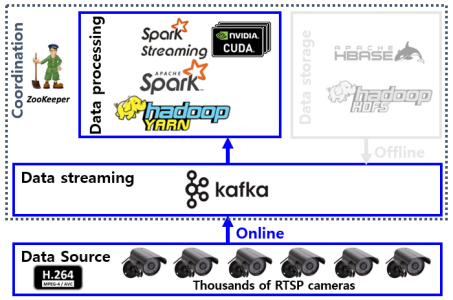
Outline

- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
    - Experiment #1: Streaming data analytics
    - Experiment #2: Static data analytics
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions



Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics
  - Purpose
    - Measure the scalability for multi-GPU & multi-node
  - Method
    - Increasing the number of clustered servers (including 1GPU each)

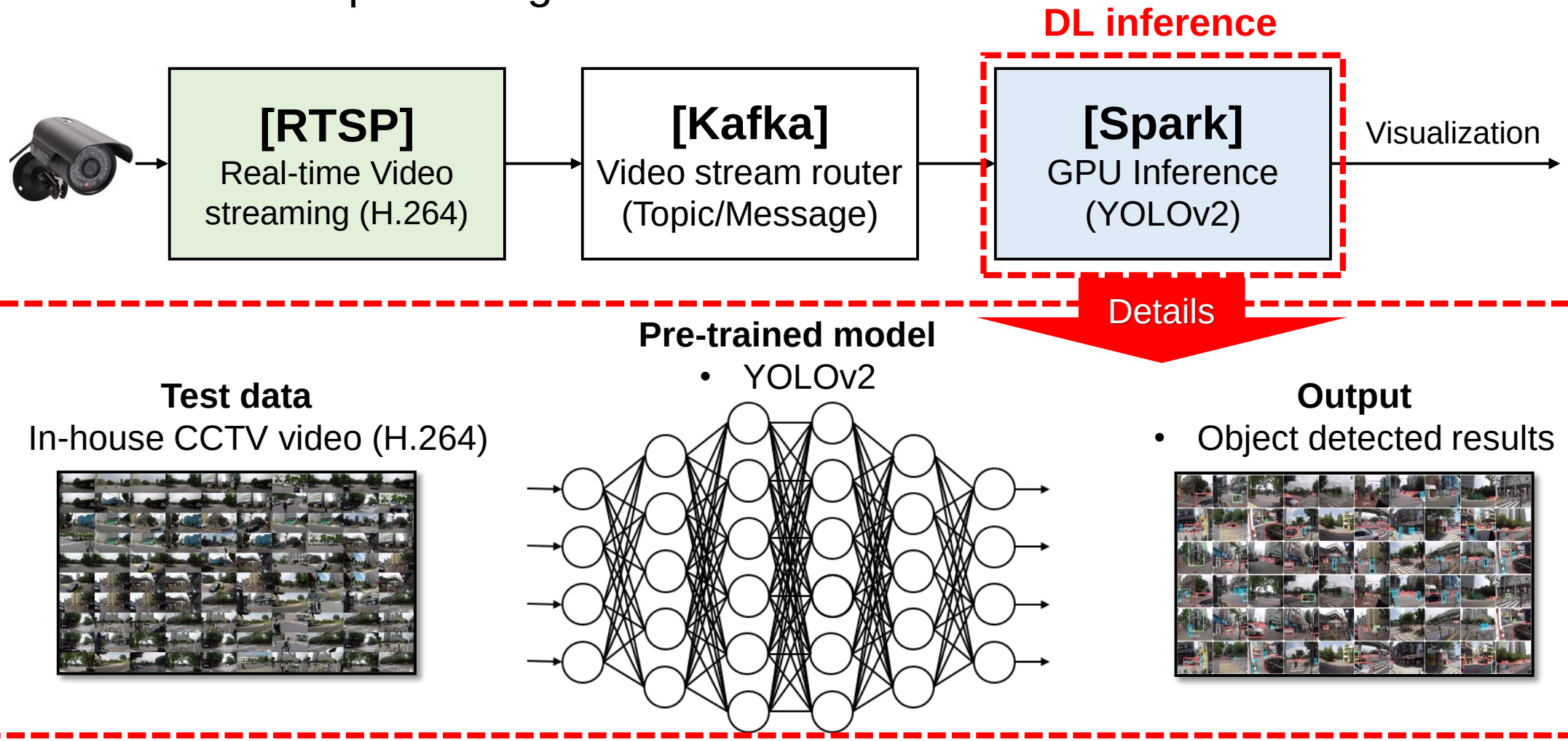


Test system	CPU	GPU	Memory	Storage	Network
31-node cluster w/ **30 GPUs**	Intel Xeon E5-2697 v4 18-cores @ 2.3 GHz	NVIDIA Tesla M60 **(1GPU/node)**	256GB DDR4	24TB HDD	10Gbps Ethernet

CentOS	CUDA	cuDNN	Tensorflow	HDP	HBase	Kafka	Spark
7.4	8.0	6.0	1.4.0-gpu	2.6.3	1.1.2	0.10.1	2.2

Solution: NVIDIA GPU & Hadoop-Spark

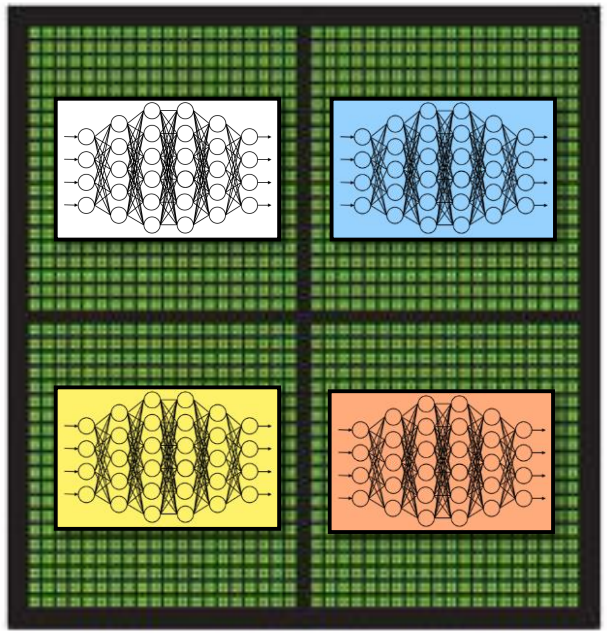
- Experiment #1: Streaming data analytics
  - Distributed deep learning inference



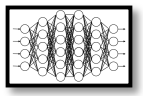
Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics
  - Distributed deep learning inference

GPU Fractional scaling



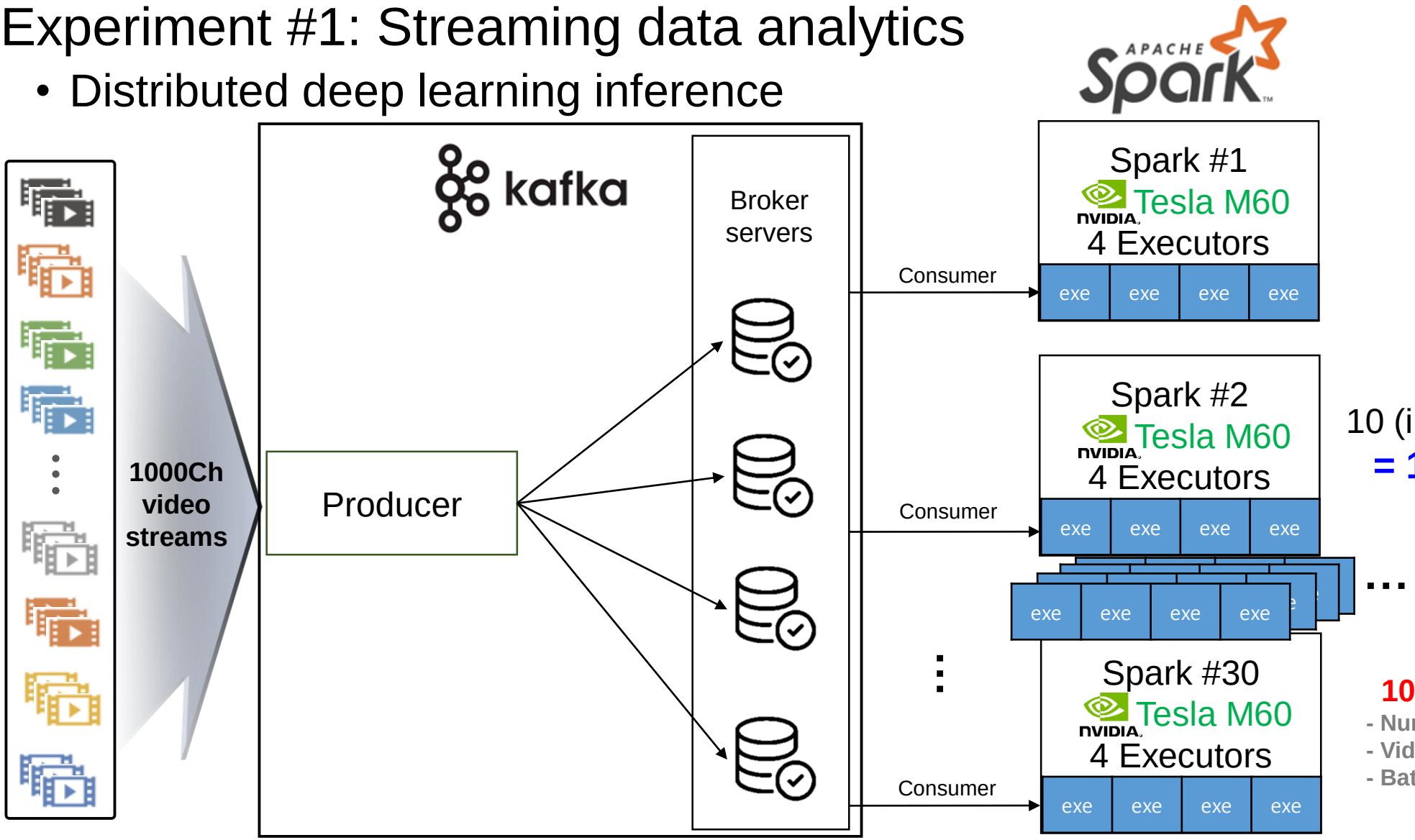
30 X 4 = 120



- **30 : # of GPUs (= # of Nodes)**
- **4 : # of YOLOv2 models in each GPU (= # of Spark executors in each GPU)**
- **120 : Total # of YOLOv2 models**

Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics
  - Distributed deep learning inference

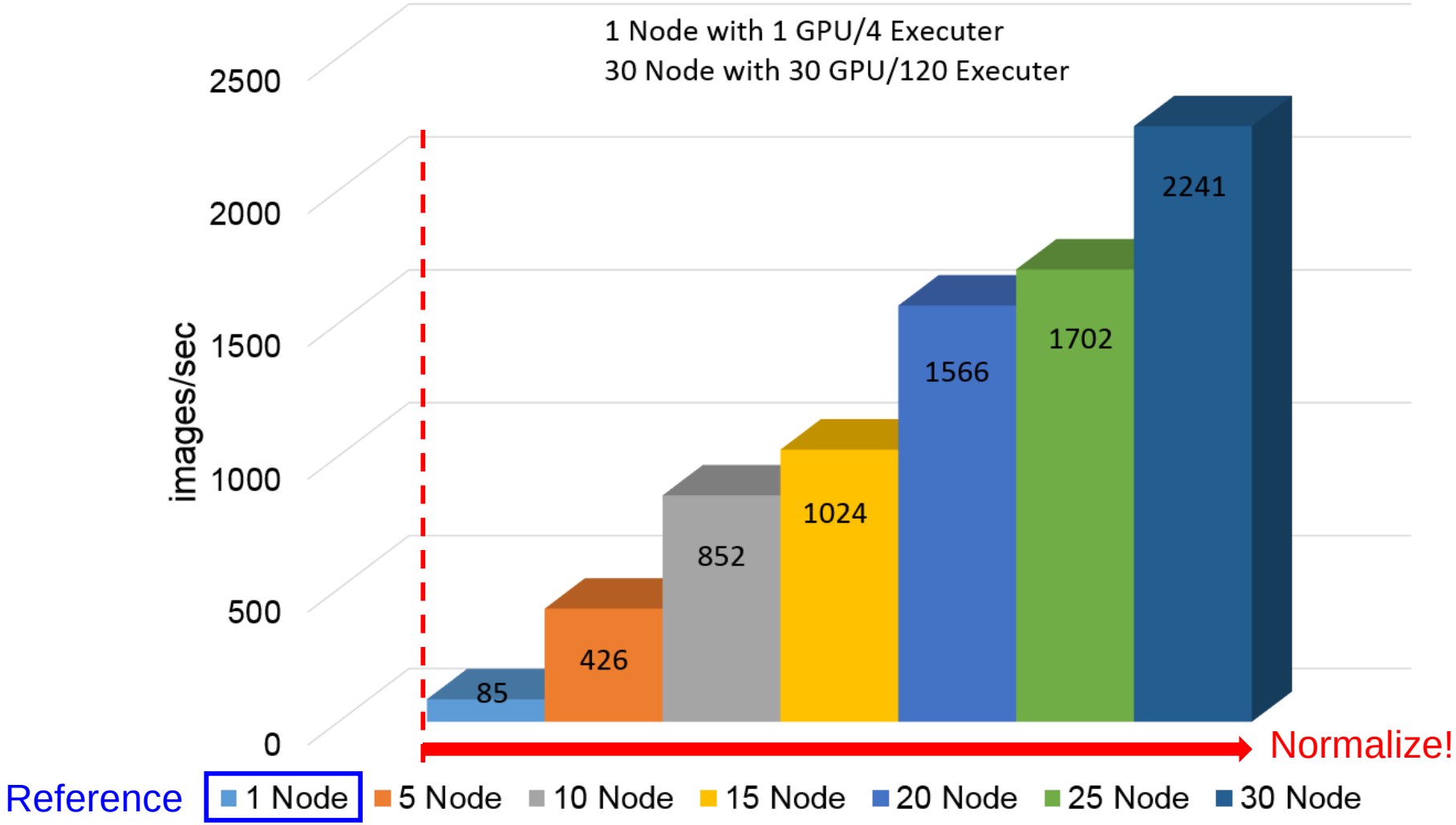


30 nodes
X
4 executor/node
X
10 (images/sec)/executor
= 1200 images/sec

1000Ch video analytics
- Number of channels: 1000 Ch
- Video decoding rate: 1 Hz
- Batch size: 1

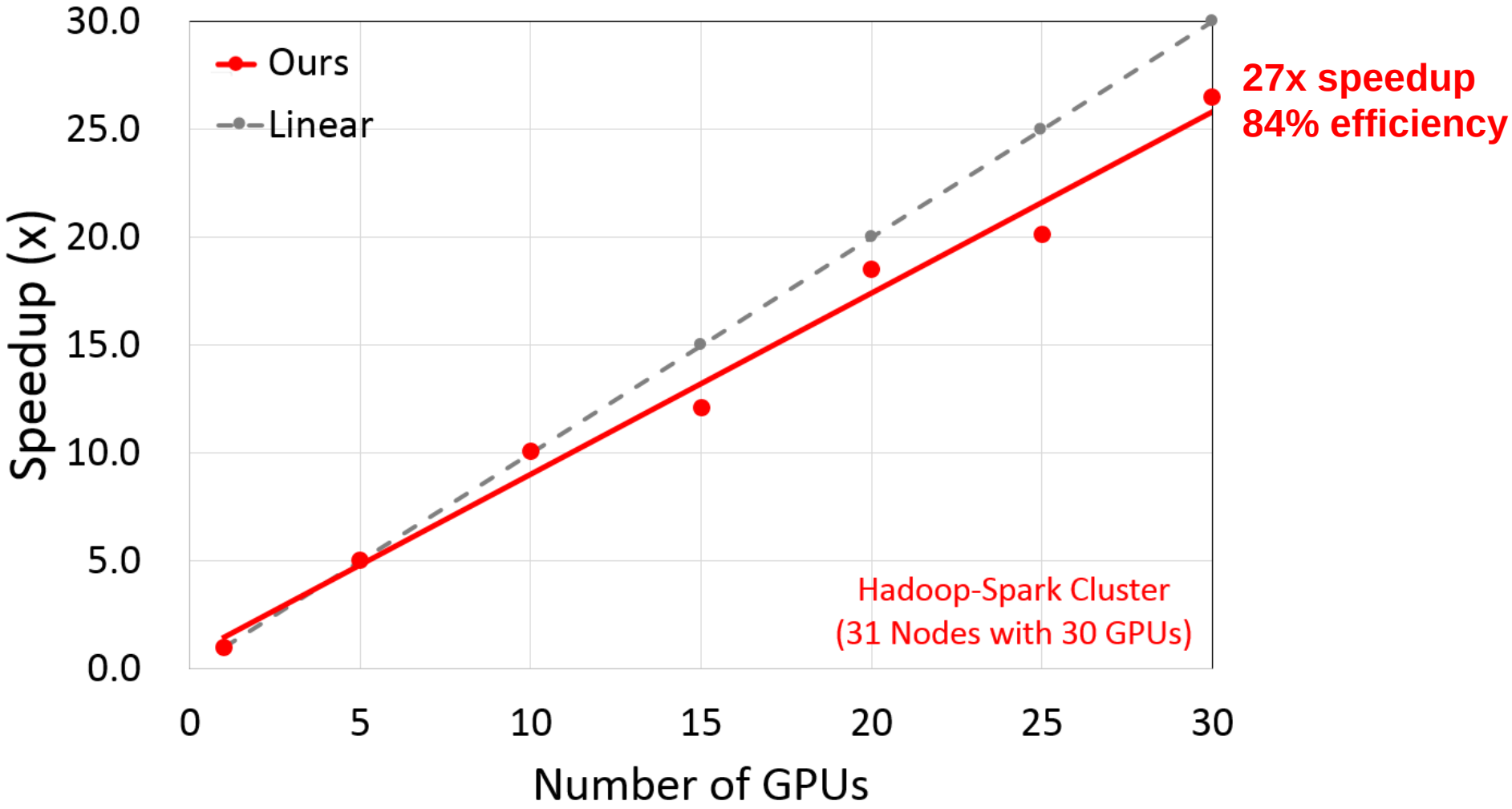
Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics
  - Results



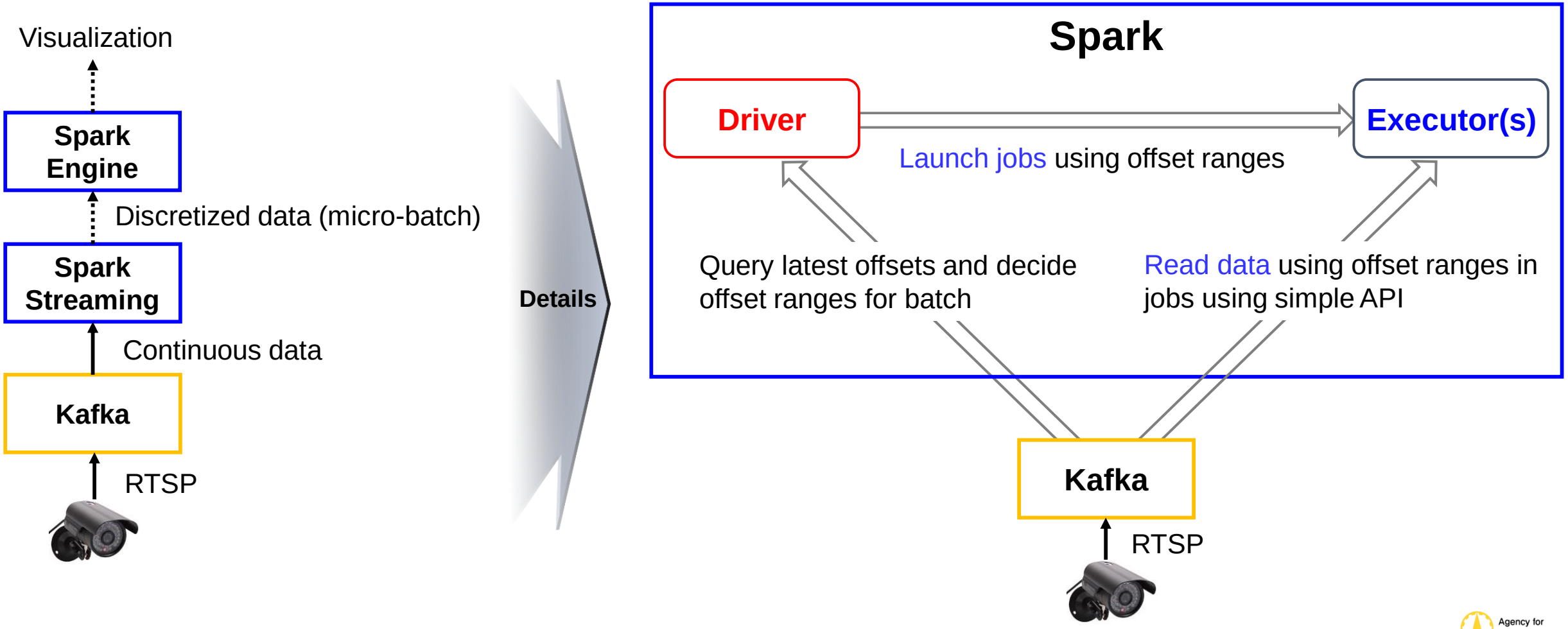
Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics
  - Results (normalized)



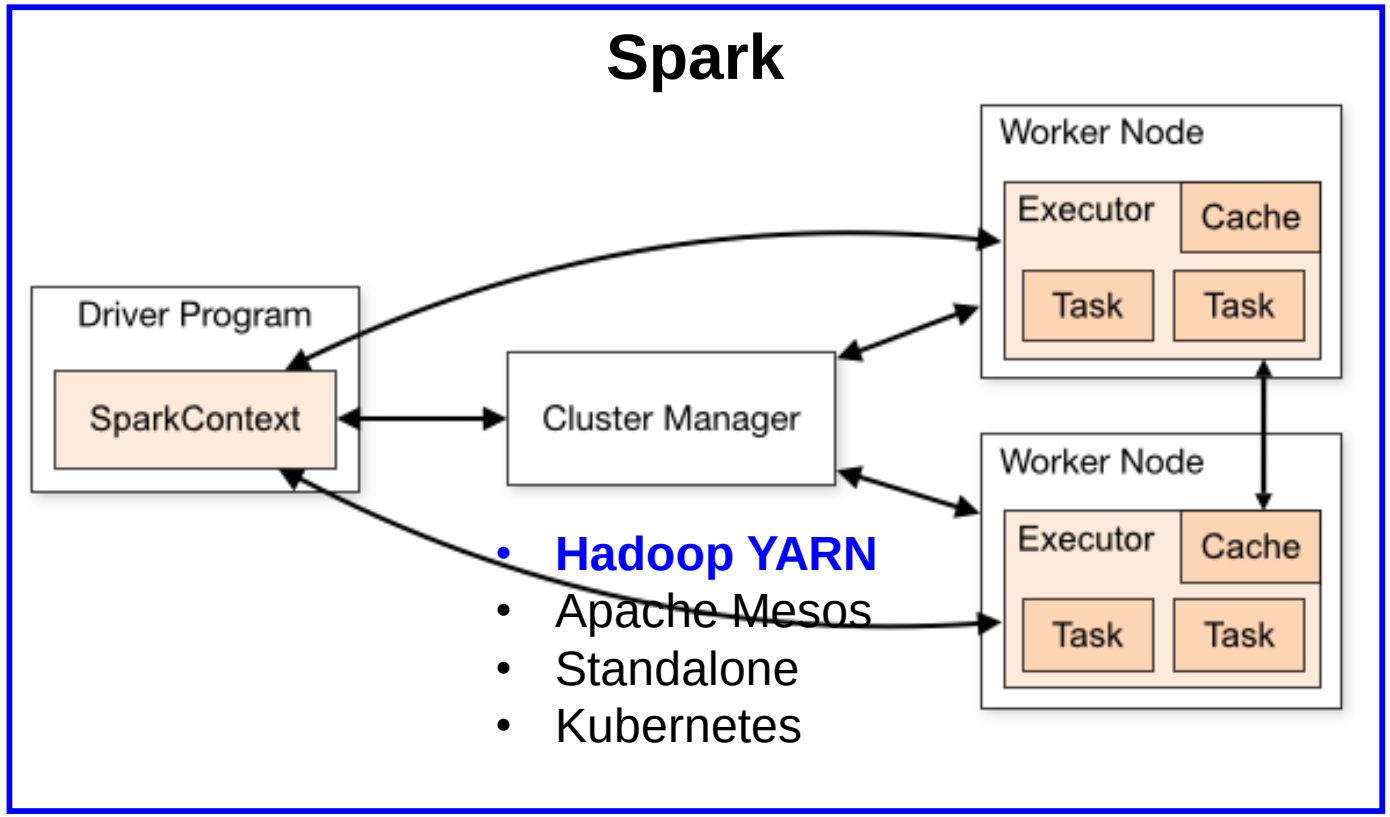
Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (more details)
  - Apache Kafka integration with Spark



Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (more details)
  - Apache Spark – ‘Cluster’ mode



Term	Meaning
Application	User program built on Spark (Driver Program + Executors)
Application jar	Jar containing user’s Spark Application
Driver program	Process running the main() of the application and creating the SparkContext
Cluster manager	External service for acquiring resources on the cluster (e.g. standalone, Mesos, YARN, etc.)
Deploy mode	Distinguishes where the driver process runs. • ‘Cluster’ mode : driver inside of cluster • ‘Client’ mode : driver outside of cluster
Worker node	Any node that can run application.
Executor	**A process** that runs tasks and keeps data in memory or disk. Each application has its own executors.
Task	**A unit of work** that will be sent to one executor
Job	A parallel computation consisting of **multiple tasks**
Stage	Each job gets divided into **smaller sets of tasks** that depend on each other

Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (examples)

Spark 2.1.1.2.6.2.0-205 | Jobs | Stages | Storage | Environment | Executors | Streaming | sparkdetect application UI

Spark Jobs (?)

A parallel computation consisting of **multiple tasks**

1 Job = 120 Tasks

User: root
Total Uptime: 20 min
Scheduling Mode: FAIR
Active Jobs: 1
Completed Jobs: 1180, only showing 1000

Event Timeline

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1180	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 (kill)	2019/03/04 17:28:54	0.8 s	0/1	0/112

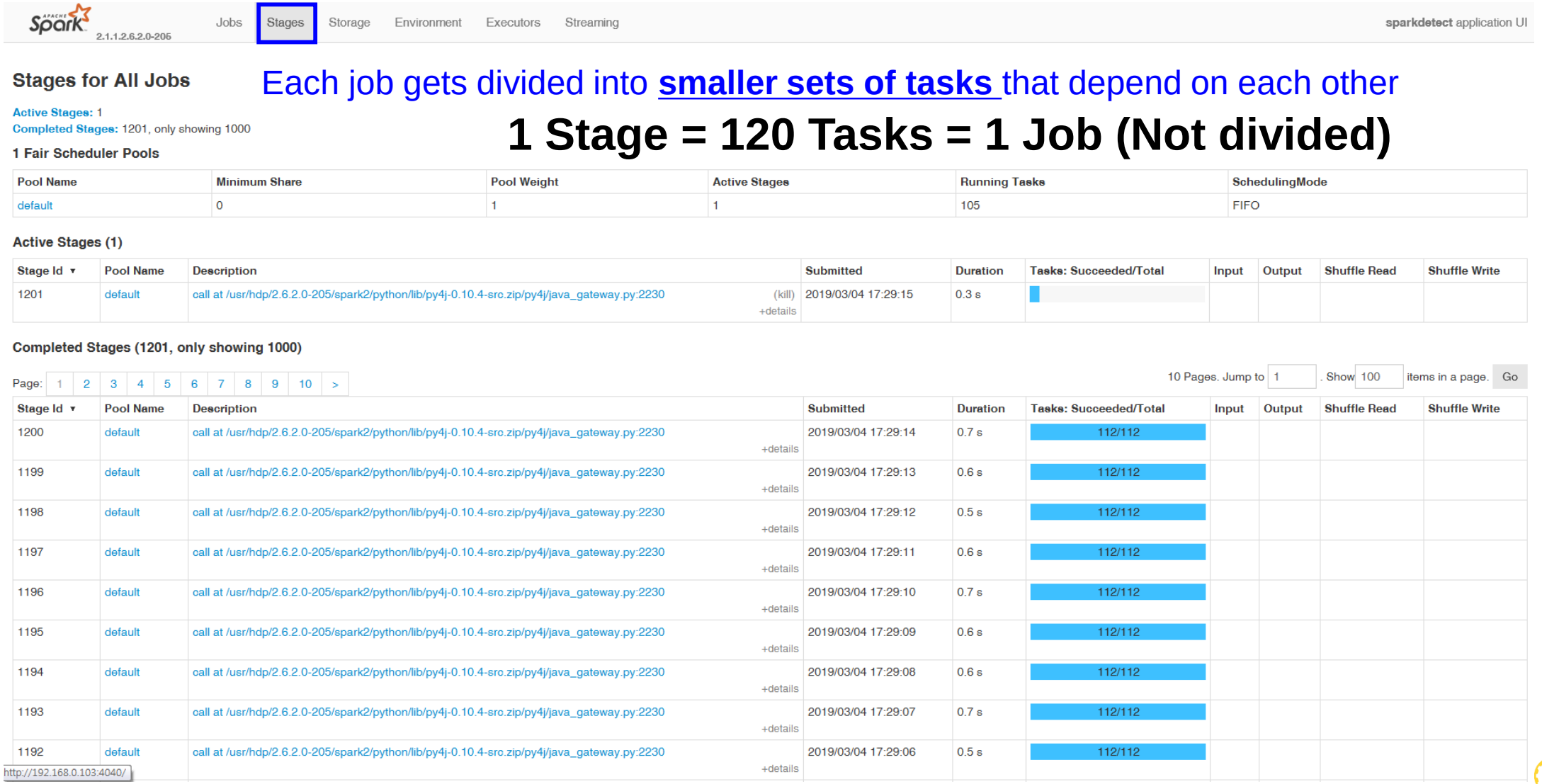
Completed Jobs (1180, only showing 1000)

Page: 1 2 3 4 5 6 7 8 9 10 > 10 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1179	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:53	0.6 s	1/1	112/112
1178	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:52	0.7 s	1/1	112/112
1177	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:51	0.7 s	1/1	112/112
1176	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:50	0.7 s	1/1	112/112
1175	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:49	0.7 s	1/1	112/112
1174	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:48	0.7 s	1/1	112/112
1173	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:47	0.7 s	1/1	112/112
1172	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:46	0.7 s	1/1	112/112
1171	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:45	0.7 s	1/1	112/112
1170	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:44	0.7 s	1/1	112/112
1169	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:43	0.7 s	1/1	112/112
1168	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:42	0.7 s	1/1	112/112
1167	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:41	0.7 s	1/1	112/112
1166	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:40	0.7 s	1/1	112/112
1165	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:39	0.7 s	1/1	112/112
1164	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230	2019/03/04 17:28:38	0.7 s	1/1	112/112

Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (examples)



Spark 2.1.1.2.6.2.0-205 | Jobs | Stages | Storage | Environment | Executors | Streaming | sparkdetect application UI

Stages for All Jobs

Each job gets divided into **smaller sets of tasks** that depend on each other

1 Stage = 120 Tasks = 1 Job (Not divided)

Active Stages: 1
Completed Stages: 1201, only showing 1000

1 Fair Scheduler Pools

Pool Name	Minimum Share	Pool Weight	Active Stages	Running Tasks	SchedulingMode
default	0	1	1	105	FIFO

Active Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1201	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 (kill) +details	2019/03/04 17:29:15	0.3 s					

Completed Stages (1201, only showing 1000)

Page: 1 2 3 4 5 6 7 8 9 10 > 10 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1200	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:14	0.7 s	112/112				
1199	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:13	0.6 s	112/112				
1198	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:12	0.5 s	112/112				
1197	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:11	0.6 s	112/112				
1196	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:10	0.7 s	112/112				
1195	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:09	0.6 s	112/112				
1194	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:08	0.6 s	112/112				
1193	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:07	0.7 s	112/112				
1192	default	call at /usr/hdp/2.6.2.0-205/spark2/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py:2230 +details	2019/03/04 17:29:06	0.5 s	112/112				

http://192.168.0.103:4040/

Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (examples)

Spark 2.1.1.2.6.2.0-205 | Jobs | Stages | Storage | Environment | **Executors** | Streaming | sparkdetect application UI

Executors

A process that runs tasks and keeps data in memory or disk.

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(113)	5763	44.6 MB / 107.5 GB	0.0 B	112	112	0	108640	108752	13.5 h (37 s)	0.0 B	0.0 B	0.0 B
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(113)	5763	44.6 MB / 107.5 GB	0.0 B	112	112	0	108640	108752	13.5 h (37 s)	0.0 B	0.0 B	0.0 B

Executors

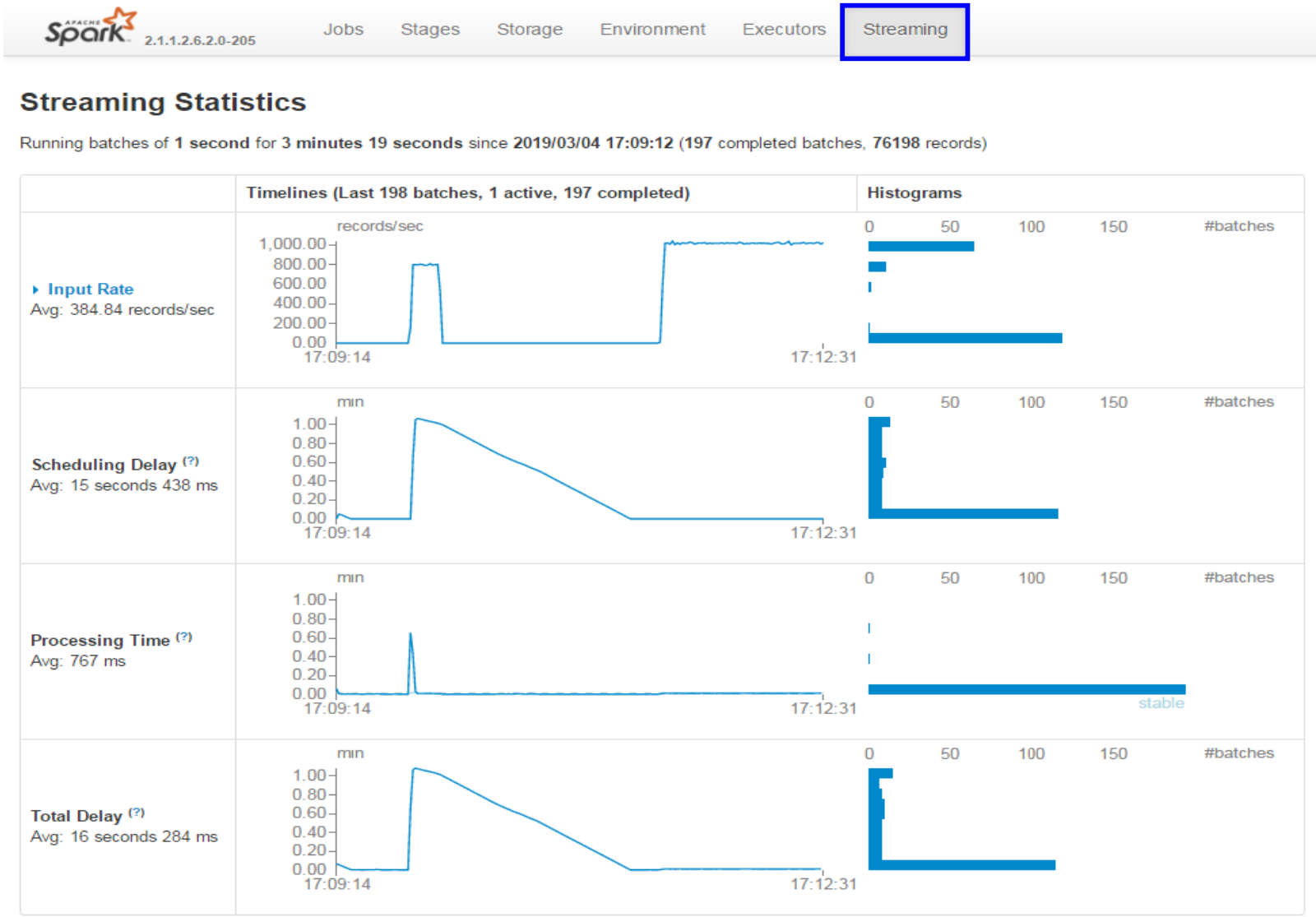
120 Executors = 120 Processes = 120 Tasks

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	192.168.0.103:38020	Active	51	394.6 KB / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	192.168.0.129:39638	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.0 min (0.3 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
1	192.168.0.129:37996	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.1 min (0.5 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
2	192.168.0.129:42337	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.2 min (0.5 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
3	192.168.0.129:37734	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.2 min (0.4 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
4	192.168.0.101:46083	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.0 min (0.4 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
5	192.168.0.101:46116	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.5 min (0.6 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
6	192.168.0.101:39364	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.6 min (0.2 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
7	192.168.0.101:40732	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.3 min (0.5 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
8	192.168.0.123:39128	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.1 min (0.4 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
9	192.168.0.123:40578	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.4 min (0.4 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
10	192.168.0.123:45122	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.4 min (0.5 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
11	192.168.0.123:41428	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.0 min (0.3 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
12	192.168.0.102:40079	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.1 min (0.5 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
13	192.168.0.102:37370	Active	51	394.6 KB / 956.6 MB	0.0 B	1	1	0	970	971	7.4 min (0.4 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (examples)



Input rate
= 1000 records/s
(1000 images/s)

Scheduling delay ≅ 0 s

Processing time < 1 s

Total delay < 1 s

Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #1: Streaming data analytics (examples)

Active

Active Batches (1)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Output Ops: Succeeded/Total	Status
2019/03/04 17:12:31	1022 records	0 ms	-	0/1	processing

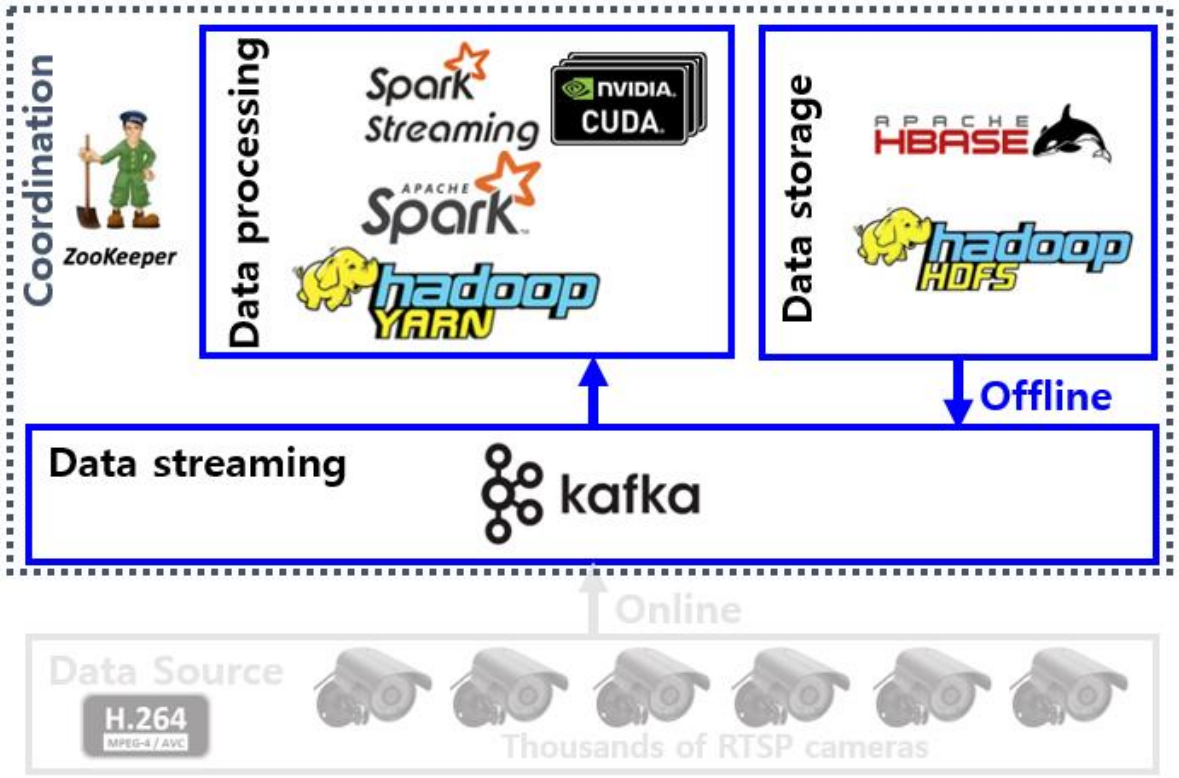
Completed

Completed Batches (last 197 out of 197)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/03/04 17:12:30	1009 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:29	1026 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:28	1021 records	1 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:27	1015 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:26	1019 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:25	1019 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:24	1012 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:23	1026 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:22	1018 records	1 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:21	1016 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:20	1017 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:19	1016 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:18	1000 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:17	1036 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:16	1020 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:15	1010 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:14	1011 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:13	1027 records	1 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:12	1024 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:11	1018 records	1 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:10	1009 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:09	1015 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:08	1015 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:07	1018 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:06	1017 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:05	1019 records	0 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:04	1015 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:03	1018 records	1 ms	0.7 s	0.7 s	1/1
2019/03/04 17:12:02	1019 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:01	1012 records	1 ms	0.8 s	0.8 s	1/1
2019/03/04 17:12:00	1016 records	0 ms	0.8 s	0.8 s	1/1
2019/03/04 17:11:59	1008 records	1 ms	0.8 s	0.8 s	1/1
2019/03/04 17:11:58	1021 records	0 ms	0.8 s	0.8 s	1/1

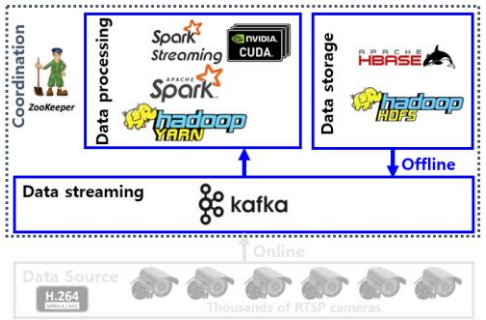
Outline

- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
    - Experiment #1: Streaming data analytics
    - Experiment #2: Static data analytics
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions

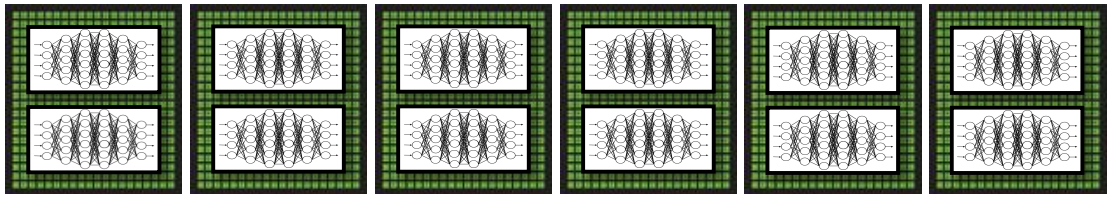


Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #2: Static data analytics
  - Purpose
    - Measure the scalability for Spark multi-processors
  - Method
    - Increasing the number of Spark executors



Multi-processing in each GPU

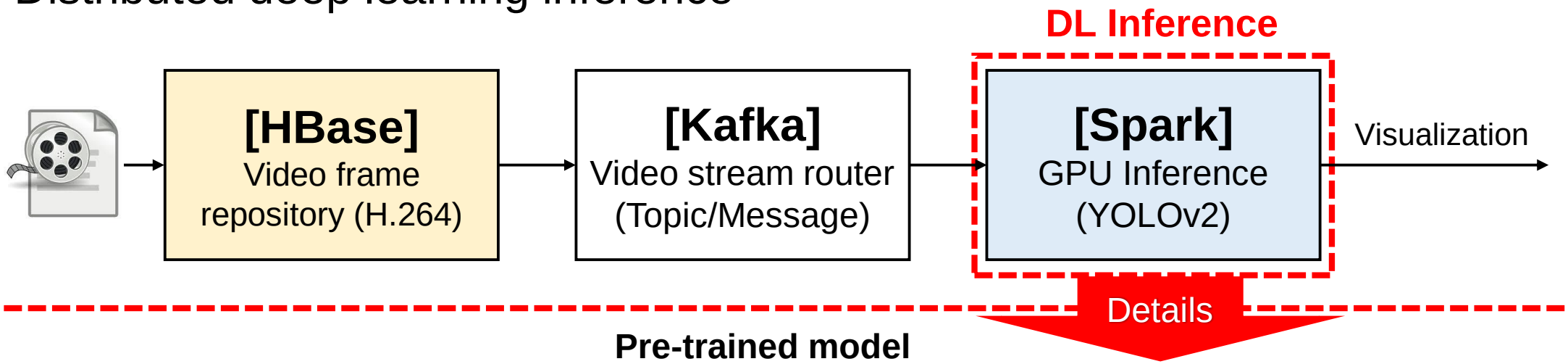


Test system	CPU	GPU	Memory	Storage	Network
4-node cluster w/ **6 GPUs**	Intel Core i7-6850K 6-cores @ 3.6 GHz	NVIDIA Titan Xp **(2GPU/node)**	32GB DDR4	1TB SSD	1 Gbps Ethernet

CentOS	CUDA	cuDNN	Tensorflow	HDP	HBase	Kafka	Spark
7.4	8.0	6.0	1.4.0-gpu	2.6.3	1.1.2	0.10.1	2.2

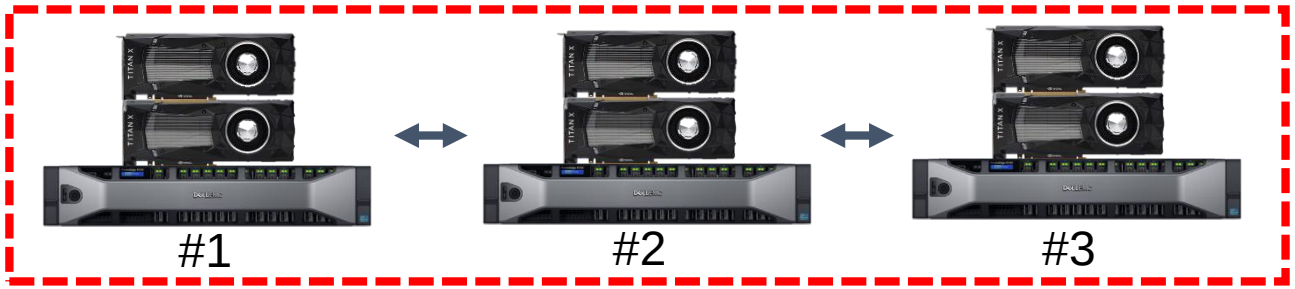
Solution: NVIDIA GPU & Hadoop-Spark

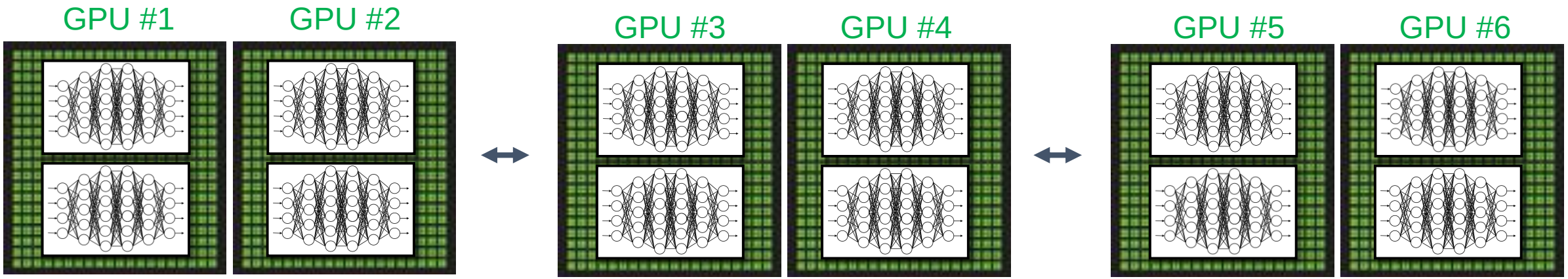
- Experiment #2: Static data analytics
  - Distributed deep learning inference



Solution: NVIDIA GPU & Hadoop-Spark

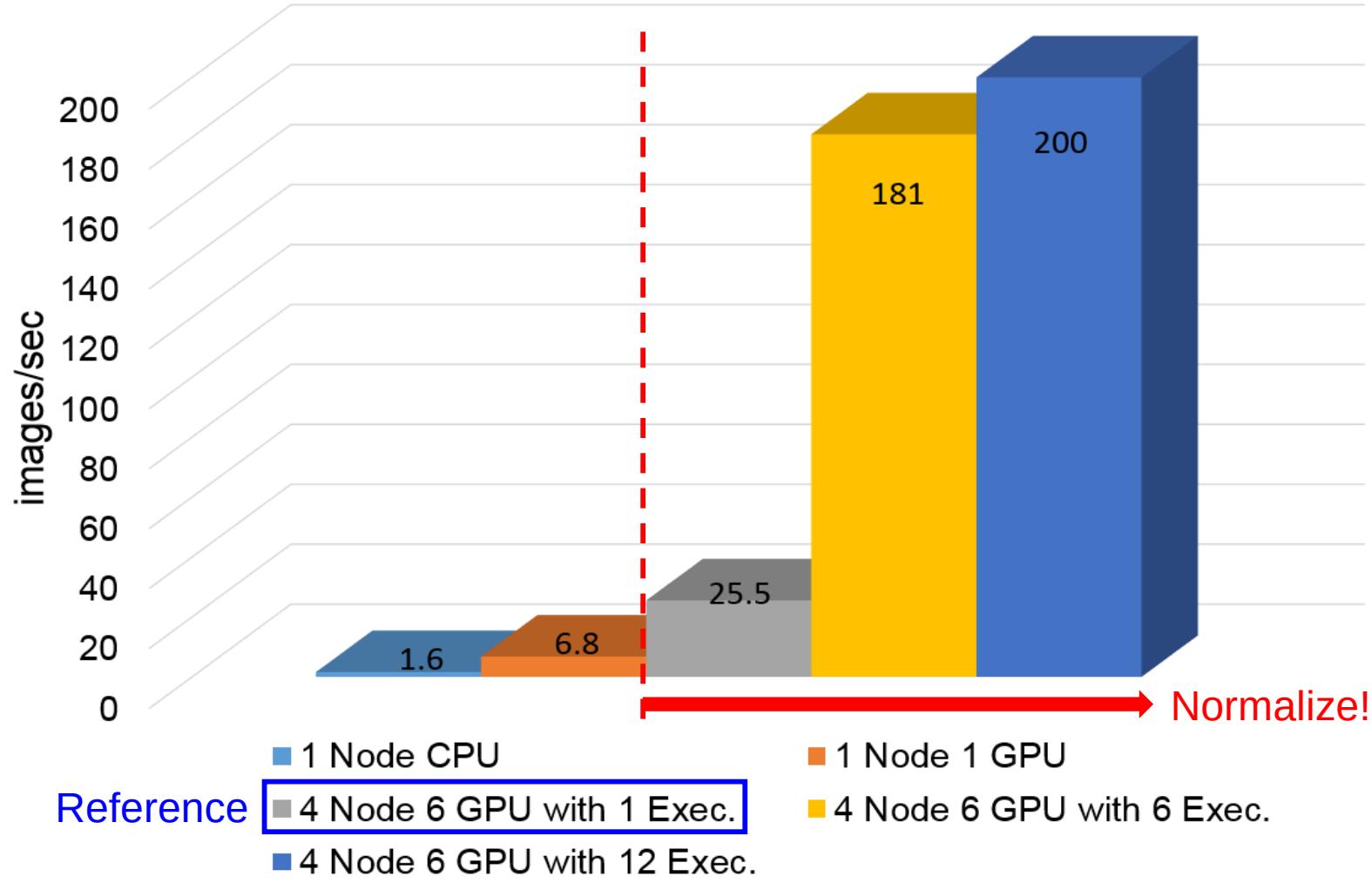
- Experiment #2: Static data analytics
  - Distributed deep learning inference





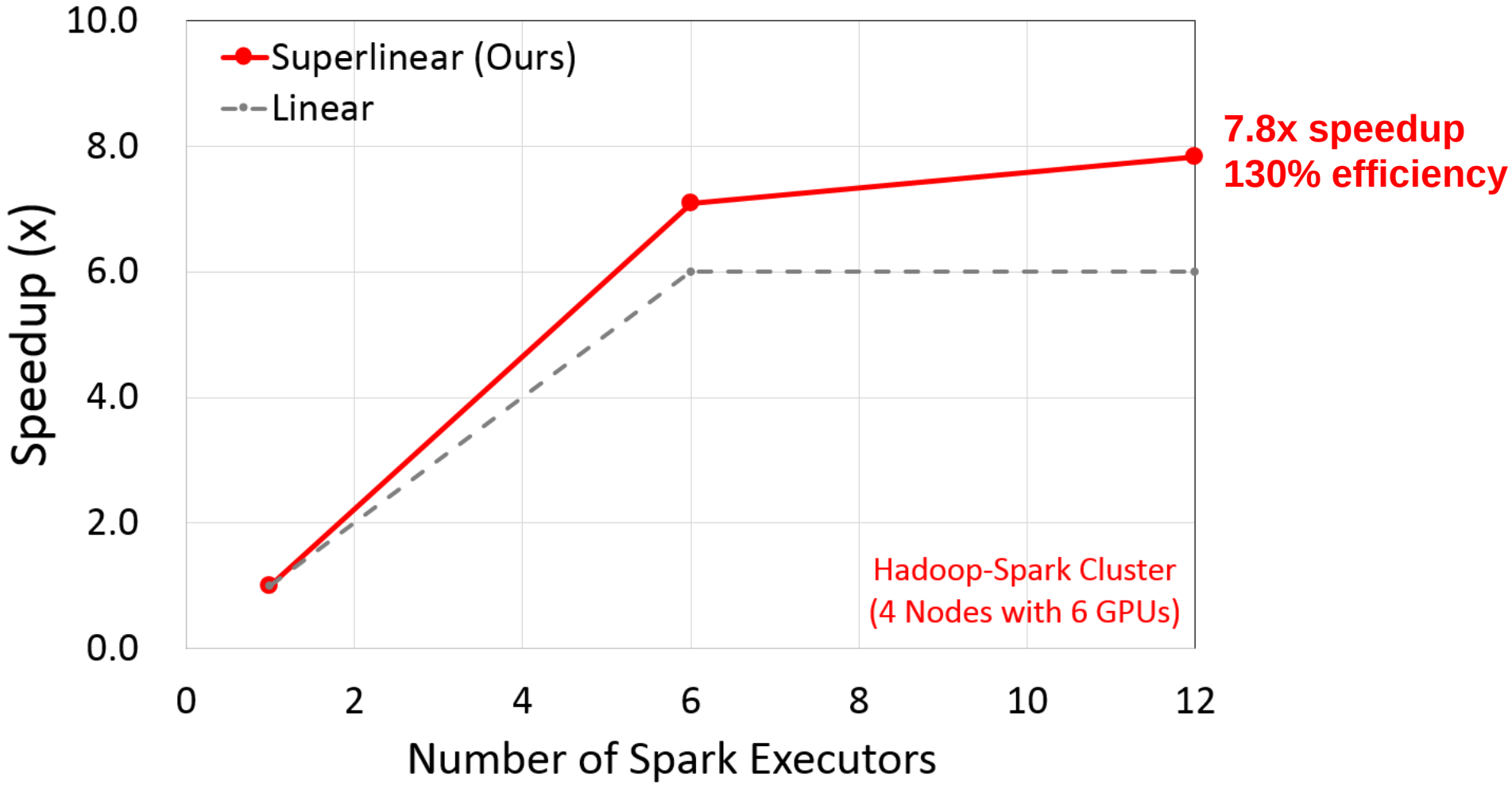
Solution: NVIDIA GPU & Hadoop-Spark

- Experiment #2: Static data analytics
  - Results



Solution: NVIDIA GPU & Hadoop-Spark

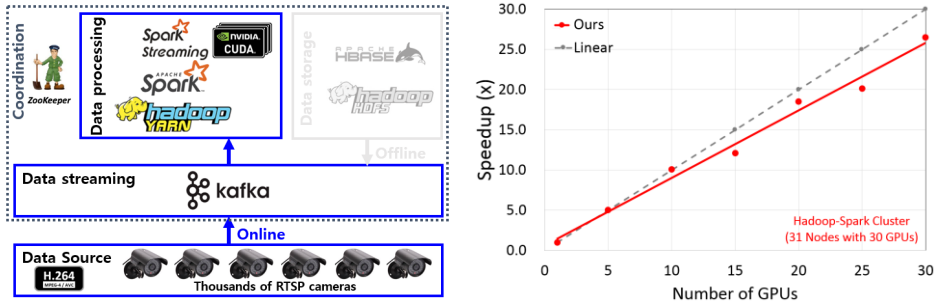
- Experiment #2: Static data analytics
  - Results (normalized)



Part 1 - Summary (Fast & Scalable DL inference)

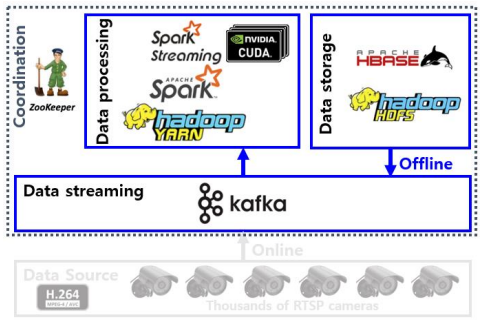
- We presented an efficient way to attain the scalable deep learning inference on multi-GPU & multi-node using Hadoop-Spark.
- We demonstrated the usage of Spark multi-processors (executors) on cluster environment to exploit the GPU resources efficiently.
- We applied the suggested approaches on both streaming and static data and achieved the high-quality scalability performance.

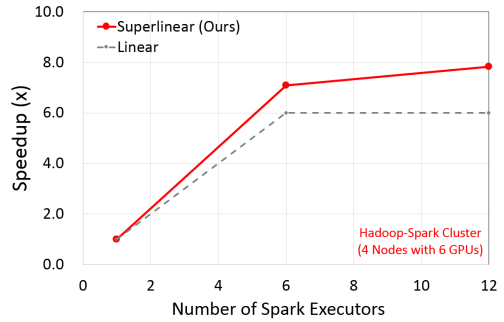
Streaming data analytics



84% efficiency

Static data analytics





130% efficiency

Part 1 - Summary (Fast & Scalable DL inference)

Single Host

[File System]
Video file repository (.mp4)

[Shell Script]
GPU Inference (YOLOv2)

Visualization

Hadoop-Spark cluster (Streaming)

[RTSP]
Real-time Video streaming (H.264)

[Kafka]
Video stream router (Topic/Message)

[Spark]
GPU Inference (YOLOv2)

Visualization

Hadoop-Spark cluster (Static)

[HBase]
Video frame repository (H.264)

[Kafka]
Video stream router (Topic/Message)

[Spark]
GPU Inference (YOLOv2)

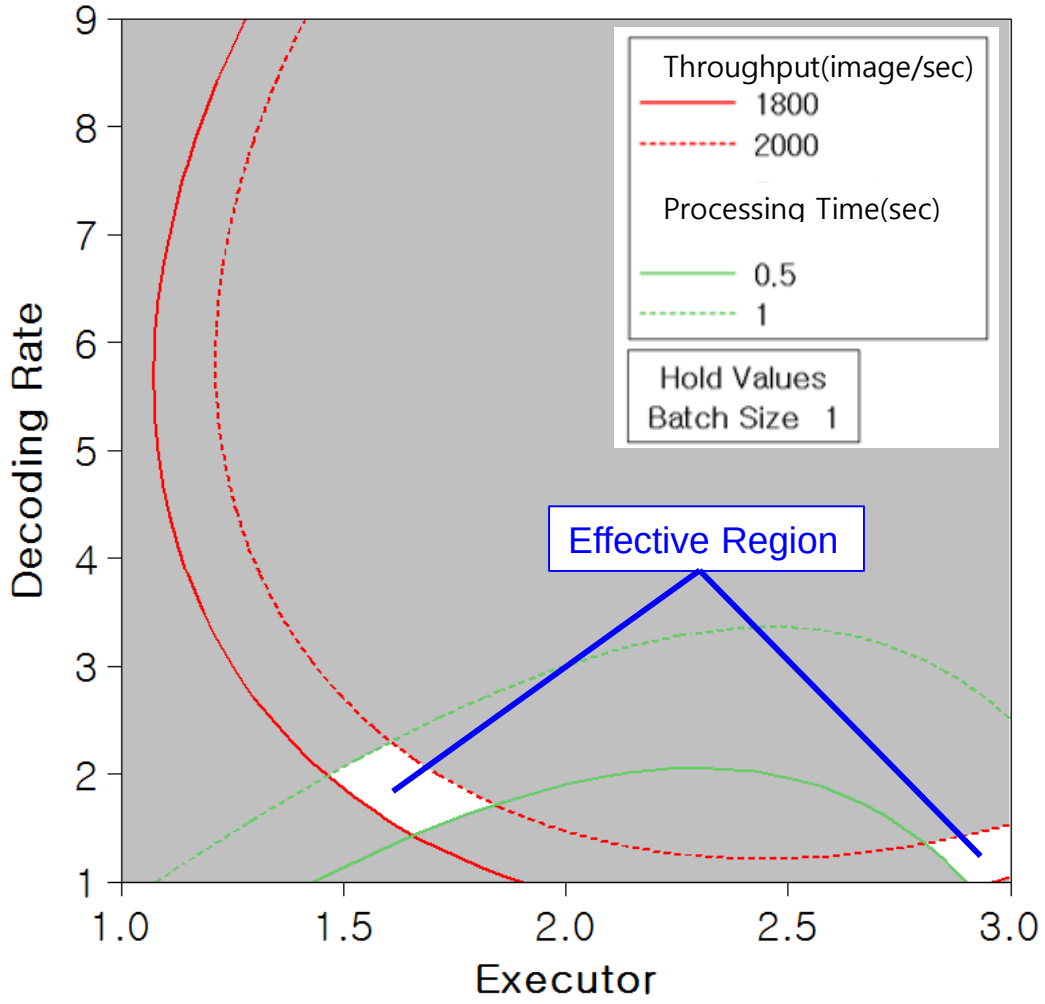
Visualization

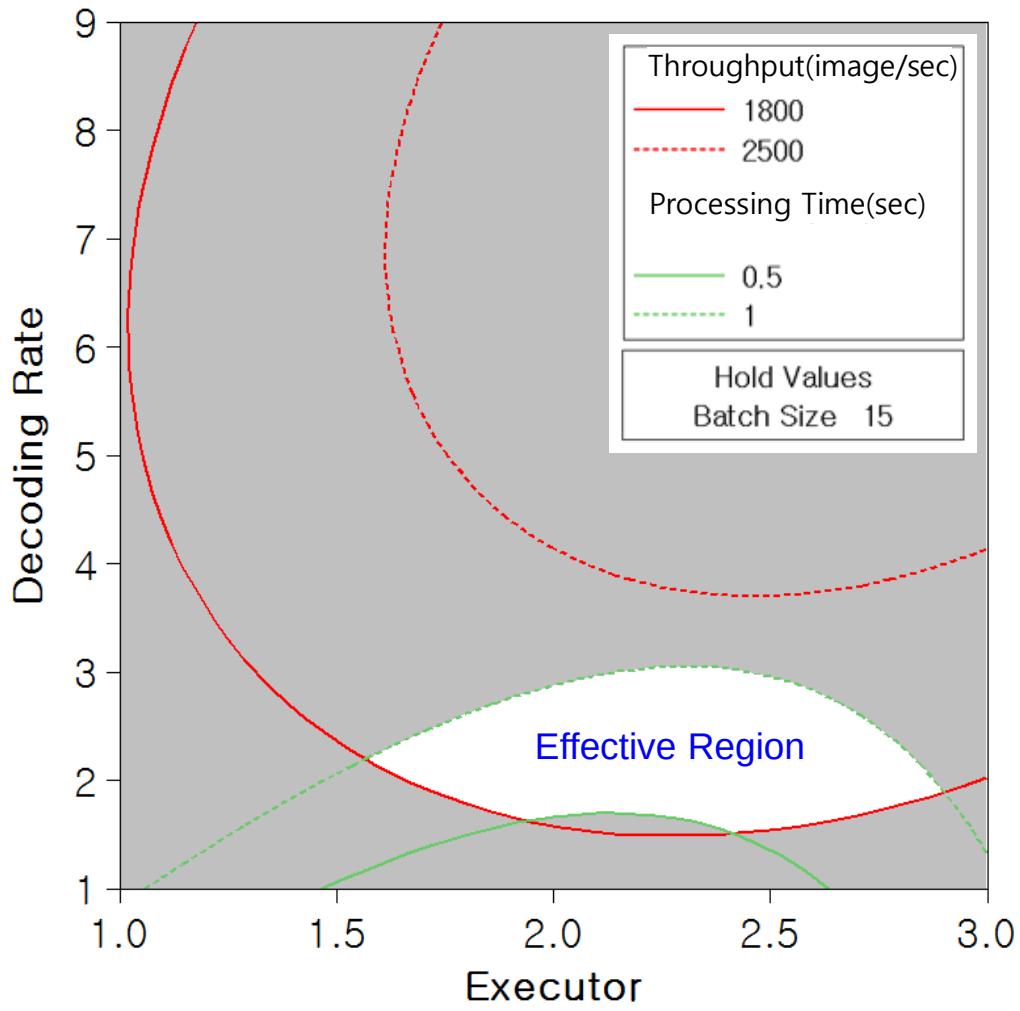
Outline

- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions

Part 2 - Overview

DL system optimization with statistical approach

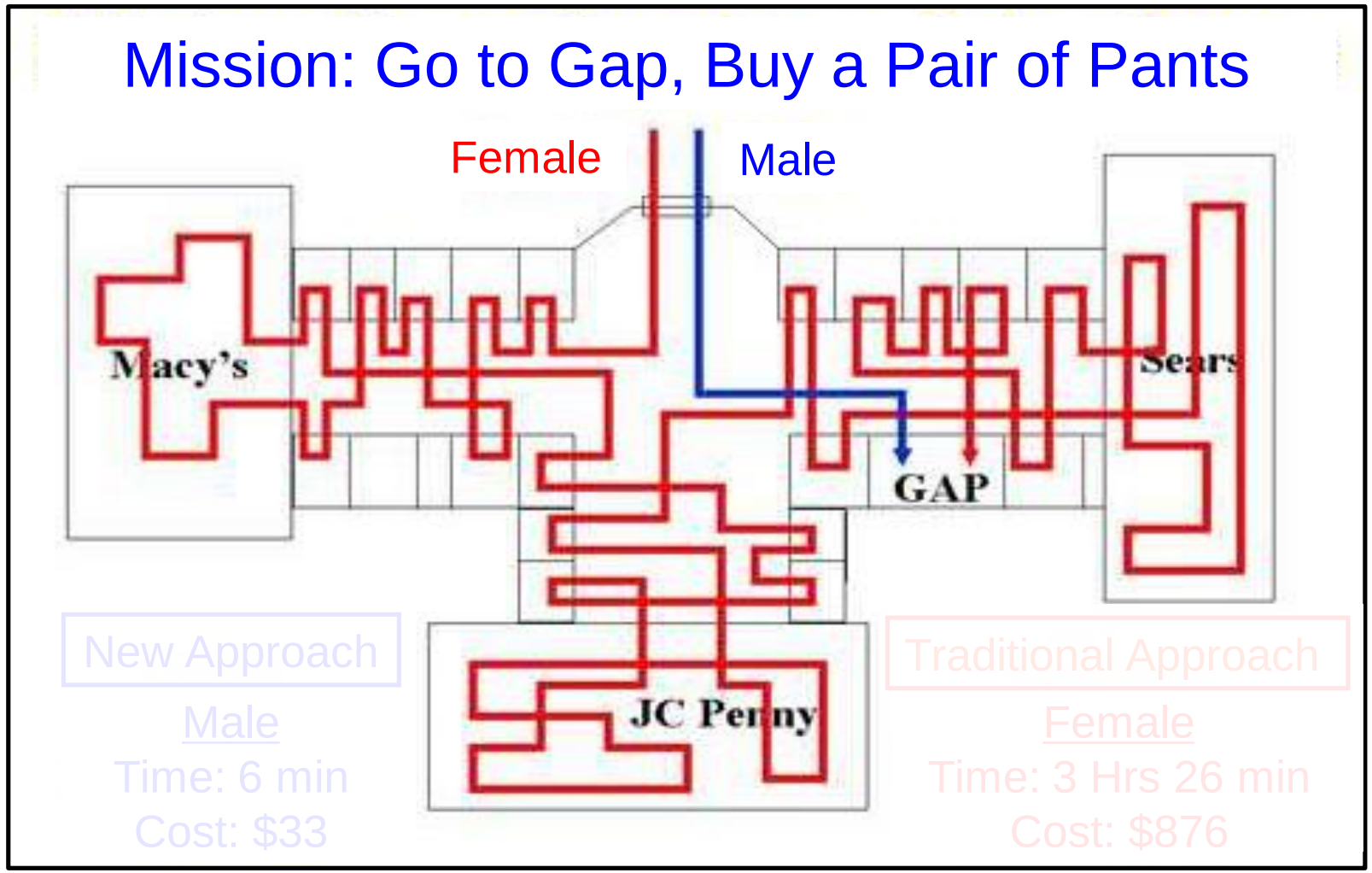




Challenge: DL System Optimization

- Good Design

How to collect useful information ?

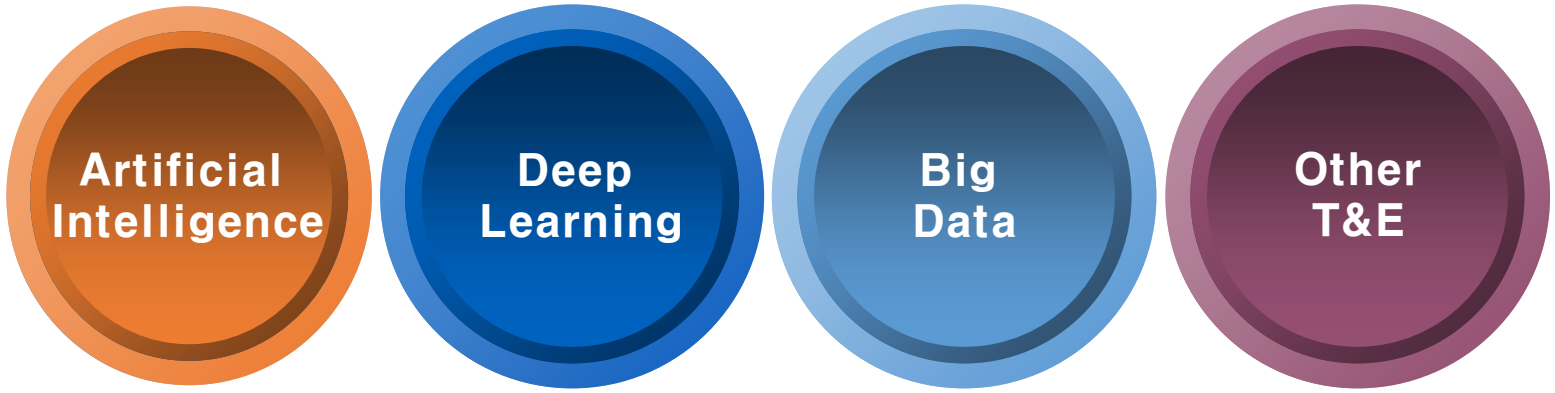


Dennis Lin @ PSU

Challenge: DL System Optimization

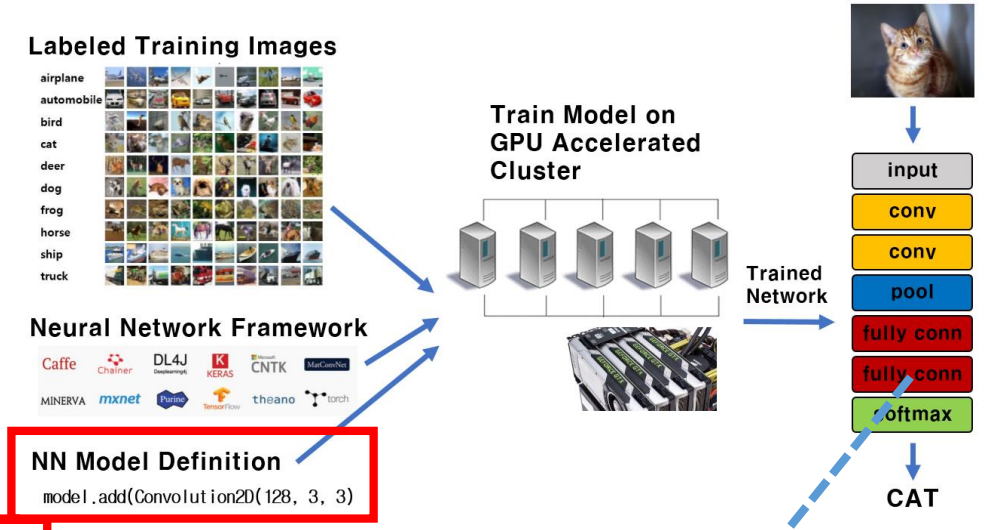
- How can we optimize the DL system?
  - Reduce design costs by speeding up the design process and reducing labor complexity.

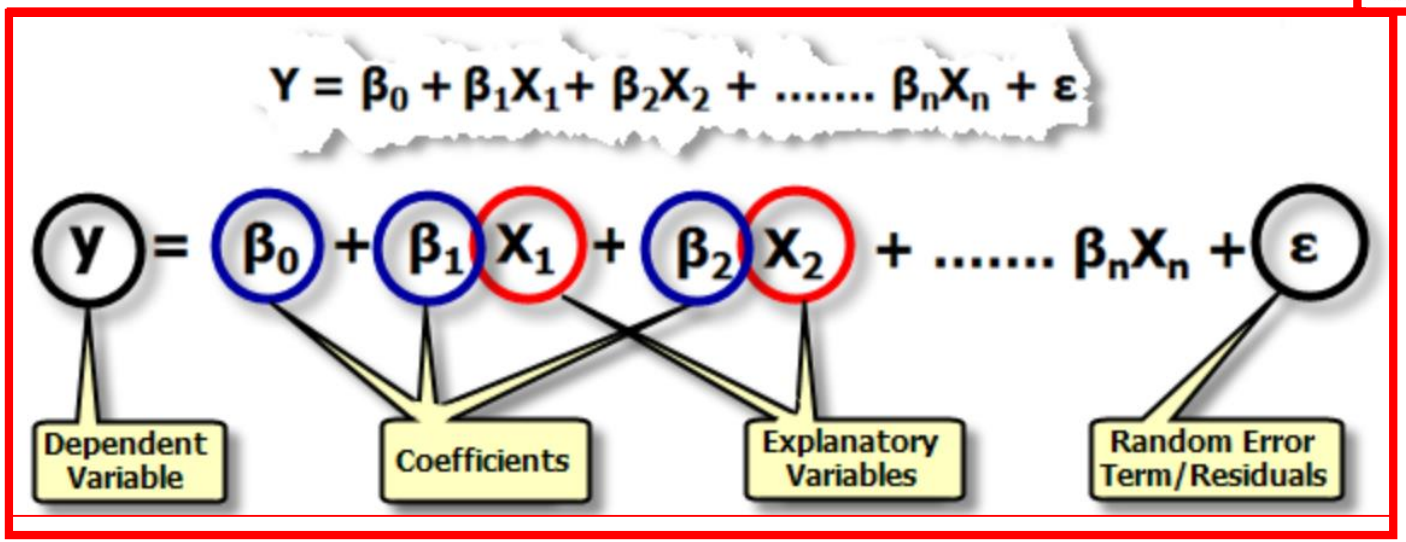
Development of Efficient R&D Methodology
Utilizing Statistical Design of Experiments (DOE)

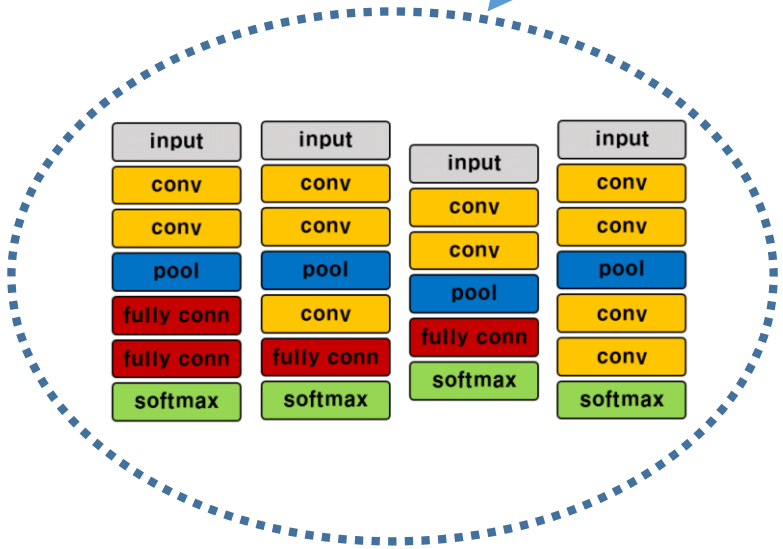


Challenge: DL System Optimization

- DL Parameter
  - DL Model Parameter
    - Labeled Data + DL Model = weight(β_n), bias(β_0)
  - Hyper-Parameter (Algorithm Design)
    - Different Learning Structure(DNN)

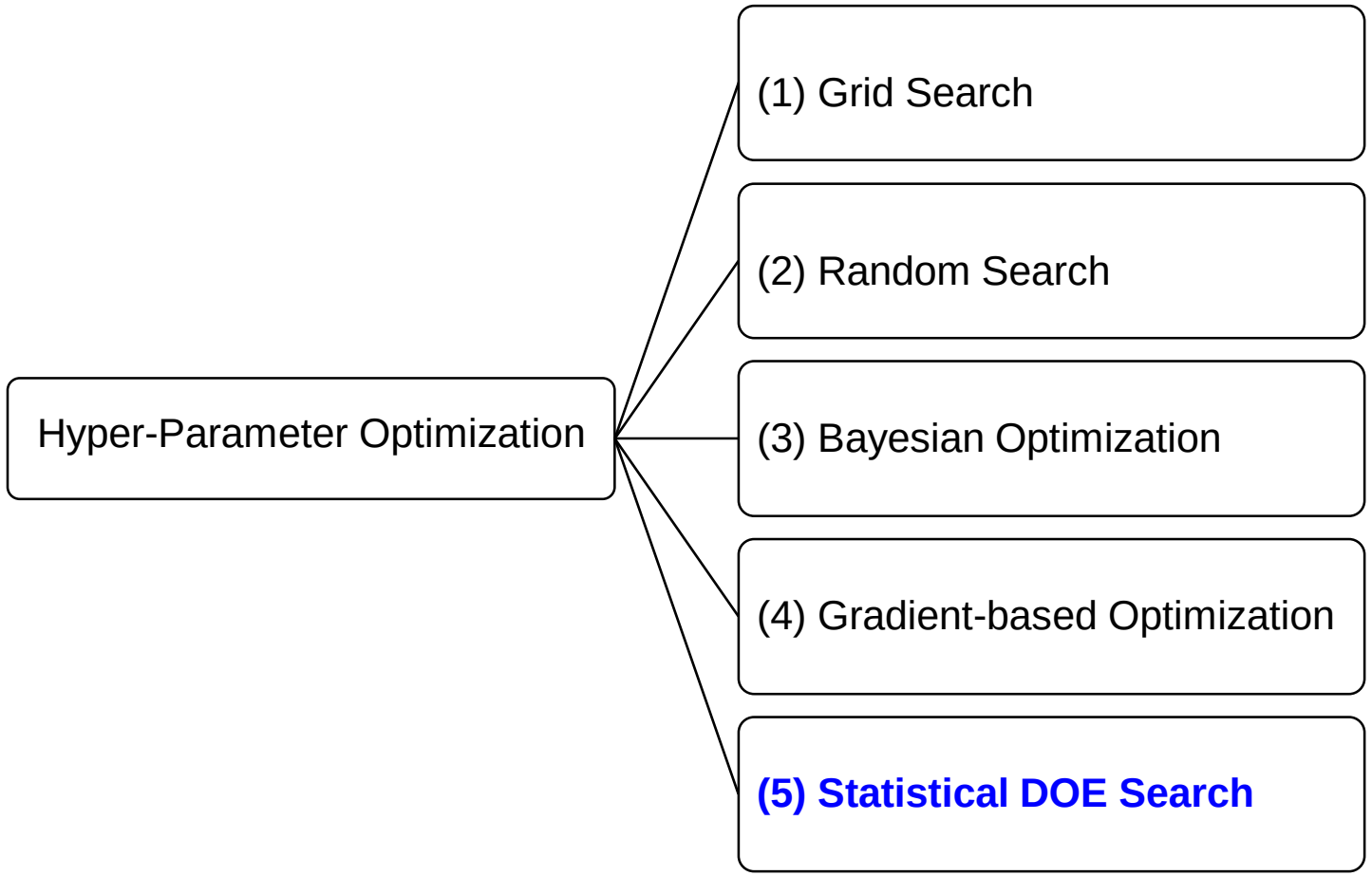






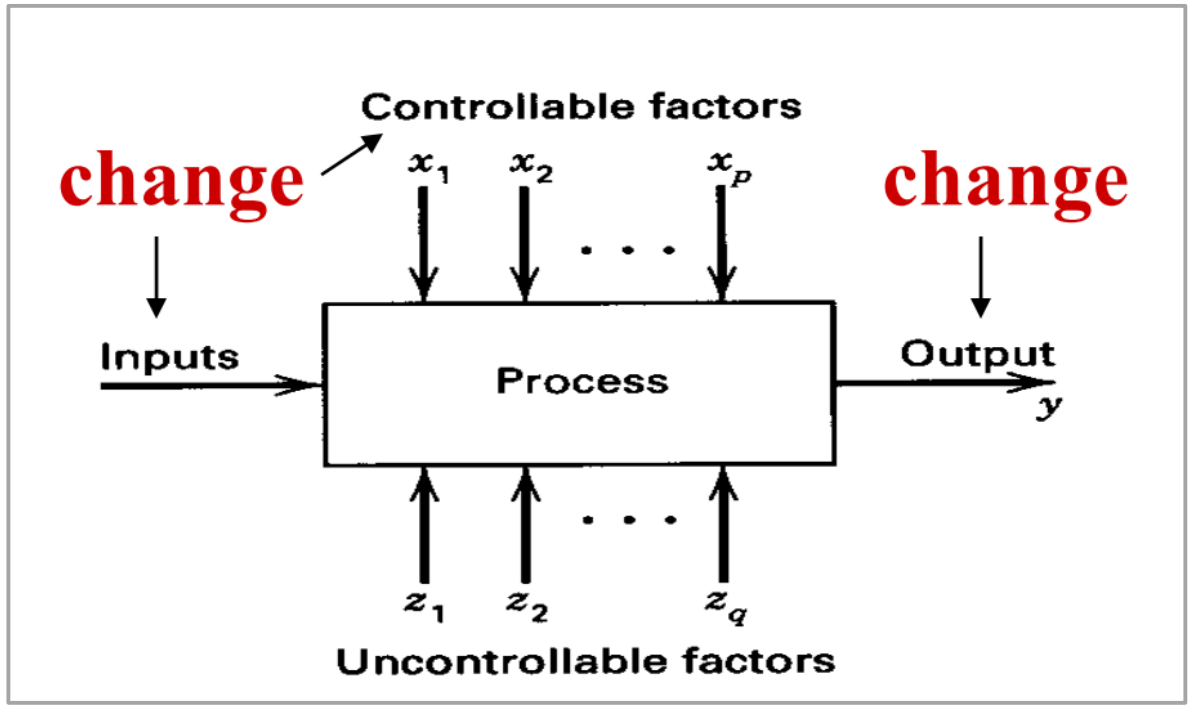
Challenge: DL System Optimization

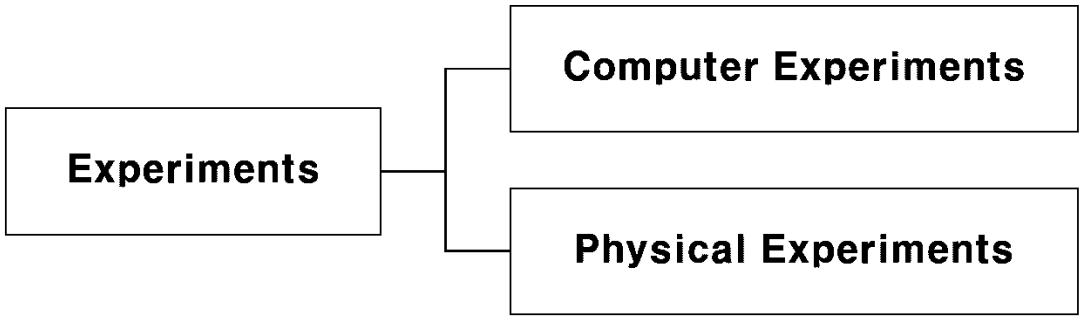
- Hyper-Parameters
  - Learning rates
  - Cost function
  - Regularization parameter
  - Batch size
  - Training epochs
  - No. of layers
  - Convolutional filters
  - Convolutional kernel size
  - Weight initialization
  - Dropout fractions
  - …



Challenge: DL System Optimization

- Experiments
  - As a test or series of tests
  - Purposeful changes are made to the input variables of system
  - Identify the reasons for the changes in the output response



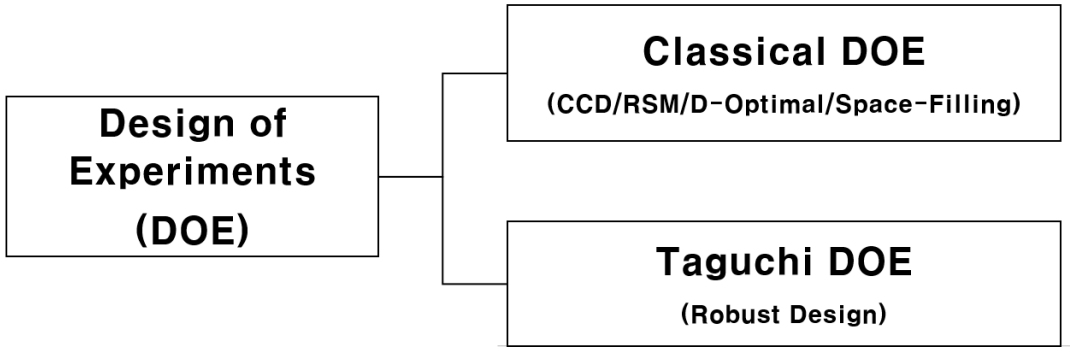


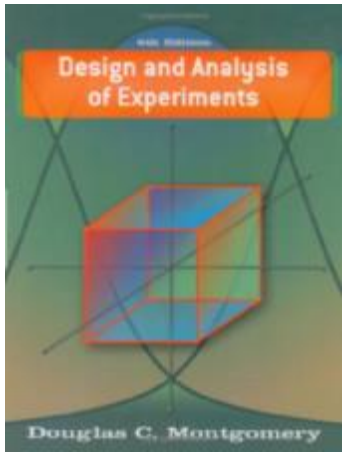
Types of experiments

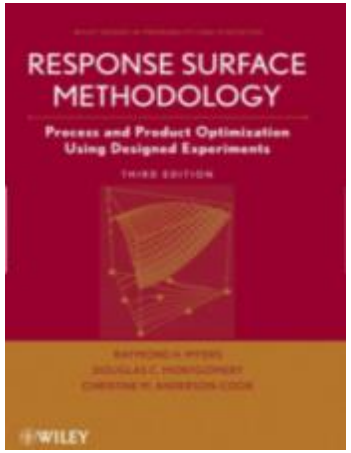
Challenge: DL System Optimization

- Strategy of Experimentation
  1. “Best-guess” experiments
     - Used a lot / Non-optimal solution with poor initial guess
  2. One-Factor-At-a Time (OFAT) experiments
     - Fail to consider any possible interaction
     - Always less efficient than statistics-based methods
  3. **Statistically designed experiments**
     - Design of Experiments (DOE)
     - Based on Fisher’s factorial concept

- Designed Experiments
- Experimental Design

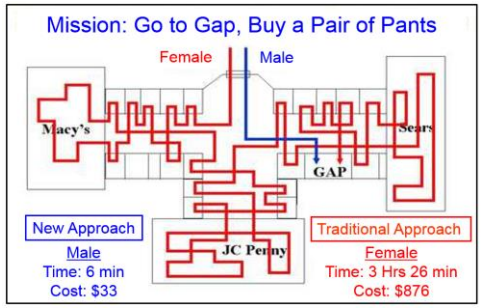


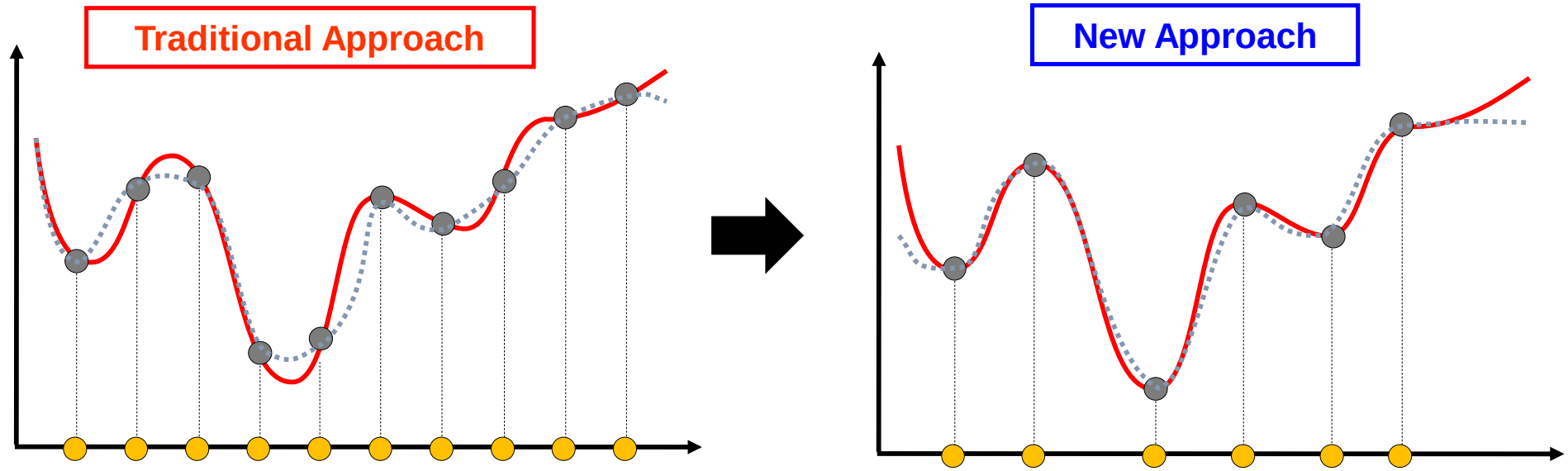




Challenge: DL System Optimization

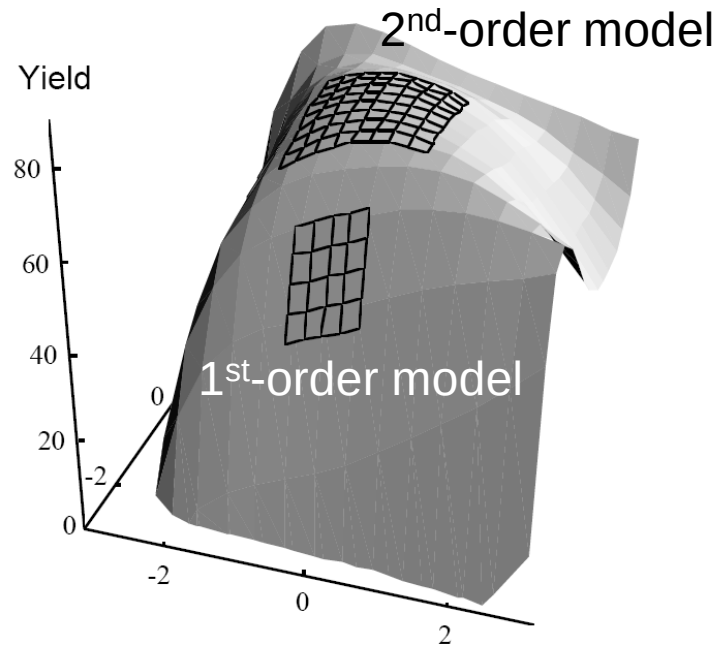
- Design of Experiments (DOE)
  - Maximum information with minimum resources
  - Develop mathematical models (factors-responses)
  - Optimize the responses
  - Mathematical response models helps in output maximization

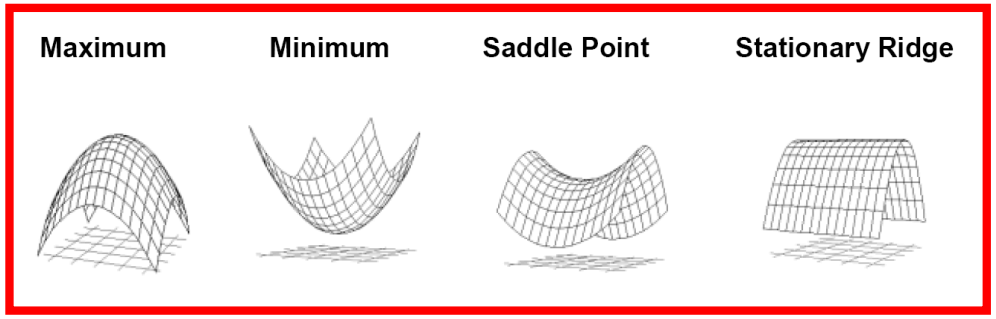




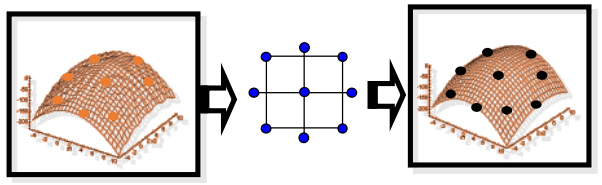
Challenge: DL System Optimization

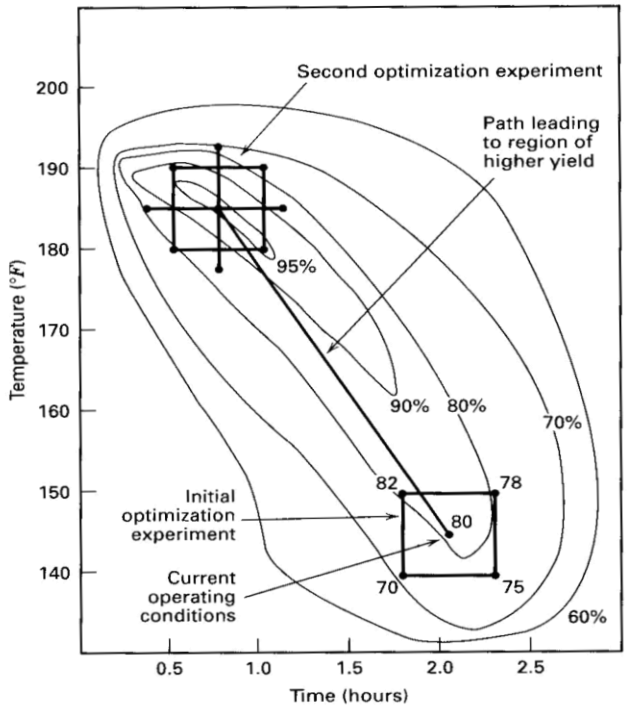
- Response Surface Methodologies (RSM/DOE)





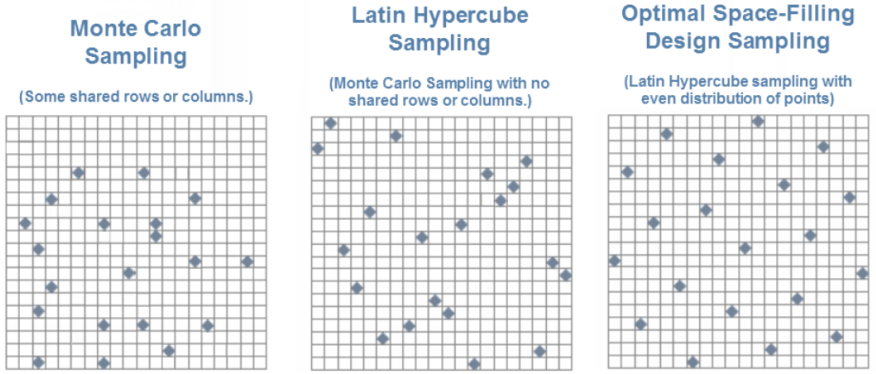
$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{ii} x^2{}_i + \sum_{i<j} \beta_{ij} x_i x_j + \varepsilon$$



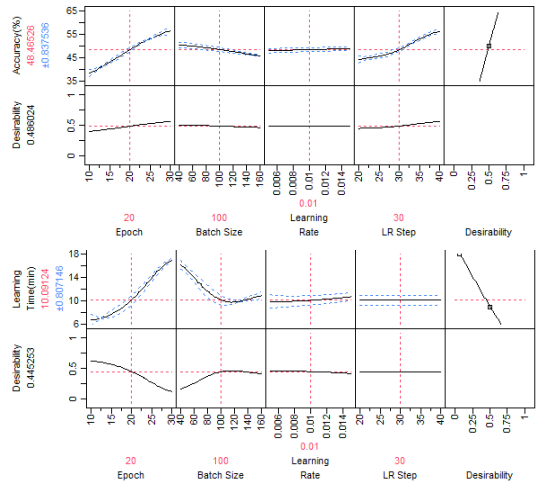


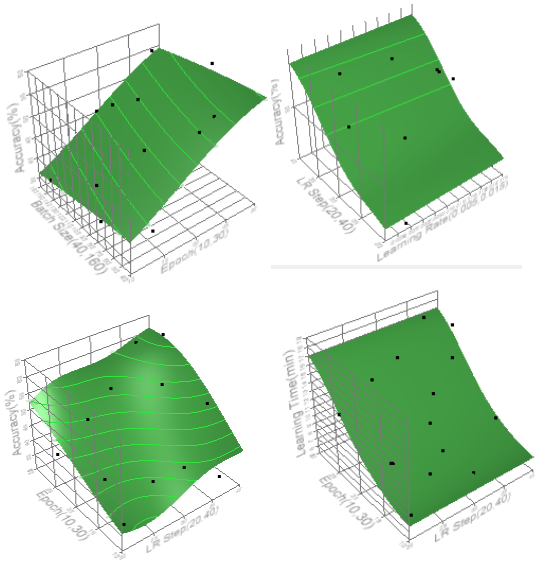
Challenge: DL System Optimization

- Case study 1: Space Filling Design
  - Cifar-100 Dataset Subset
    - 20 categories
    - 50,000 color images
  - Factors as design variables
    - No. Epoch(X1)
    - Batch Size(X2)
    - Learning Rate(X3)
    - LR Step(X4)
  - Responses
    - Accuracy(Y1)
    - Learning Time(Y2)



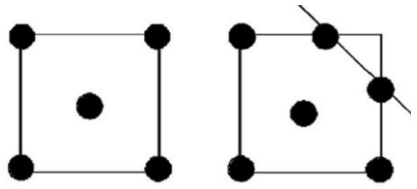
RUN No	Epoch(x1)	Bach Size(x2)	Learning Rate(x3)	LR Step(x4)	Accuracy(%)	Time(min)
1	22	122	0.014	33	51.6	12
2	24	147	0.007	20	43.0	13
3	21	46	0.007	35	55.6	16
4	13	40	0.011	27	40.1	11
5	29	91	0.009	37	59.4	17
6	14	109	0.005	26	38.6	8
7	17	135	0.012	23	40.6	9
8	25	59	0.013	29	54.4	16
9	12	97	0.010	32	41.9	7
10	28	154	0.011	34	55.6	15
11	15	65	0.013	38	52.6	10
12	26	103	0.006	28	52.8	14
13	11	84	0.009	21	36.1	7
14	27	78	0.014	40	61.7	16
15	23	53	0.008	24	51.3	16
16	16	72	0.015	22	42.2	10
17	19	160	0.008	31	45.3	10
18	30	116	0.012	25	55.5	16
19	18	128	0.010	39	51.7	9
20	10	141	0.006	36	37.3	5

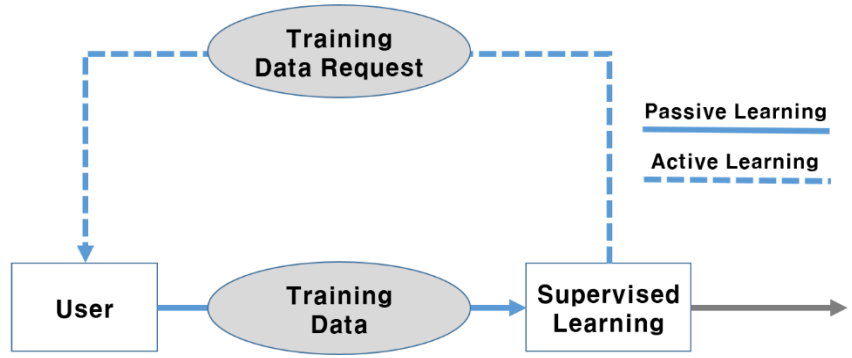


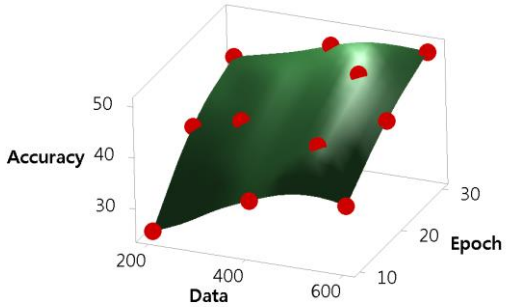


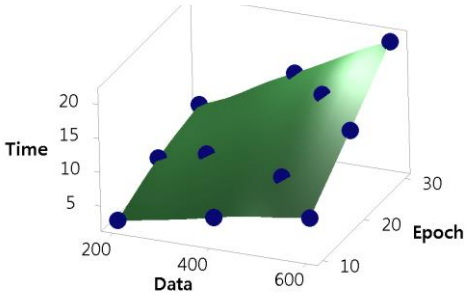
Challenge: DL System Optimization

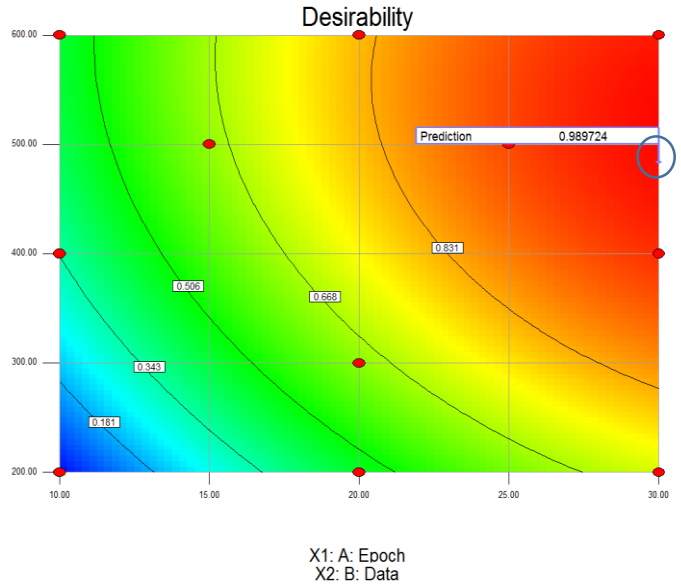
- Case study 2: D-Optimal Design
  - Cifar-100 Dataset Subset
    - 20 categories
    - 50,000 color images
  - Factors as design variables
    - No. Epoch(X1)
    - Dataset Size(X2)
  - Responses
    - Accuracy(Y1)
    - Learning Time(Y2)
  - Optimum Results
    - Active learning to help in dataset selection
    - Accuracy : 50 % @ min. Learning Time





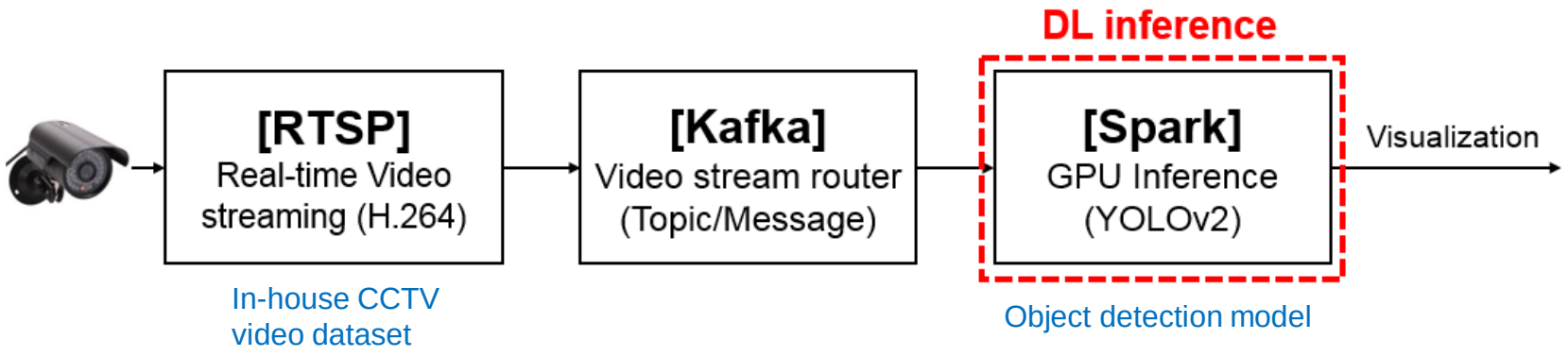


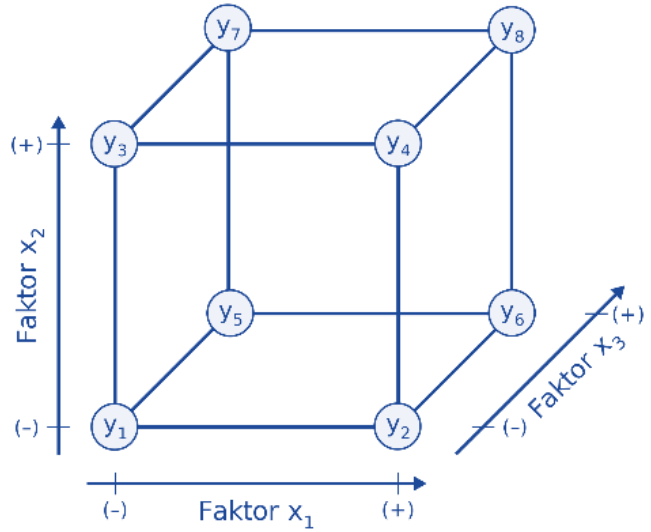




Solution: the statistical DOE based system optimization

- System design parameters
  - X1 : Number of Spark executors per GPU
  - X2 : Video decoding rate (Hz)
  - X3 : Batch size for DL model inference
- Responses (Performance metrics)
  - Y1 : Throughput in processed images per second (images/sec)
  - Y2 : Processing time (sec)





Solution: the statistical DOE based system optimization

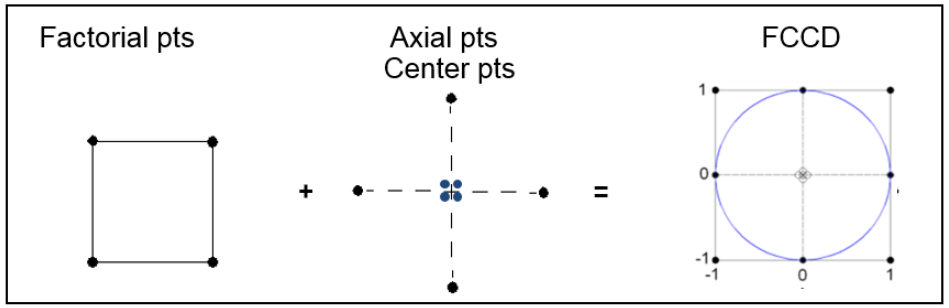
- FCCD experiments in our solution
  - Special case of ordinary CCD
  - Second-order regression model
    - very flexible / wide variety function form
    - easy estimation / work well in real problems

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{ii} x_i^{\,2} + \sum_{i=1}^{k-1} \sum_{j=2}^{k} \beta_{ij} x_i x_j + \varepsilon$$

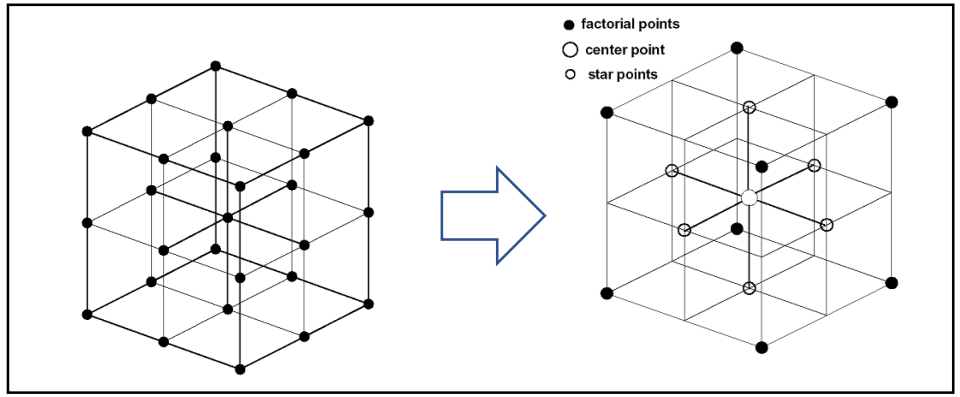
where $E(\varepsilon) = 0$

$$Var(\varepsilon) = \sigma^2$$

$\{\varepsilon\}$ are uncorrelate random variables



2 variable FCCD



3 variable factorial design

3 variable FCCD

Solution: the statistical DOE based system optimization

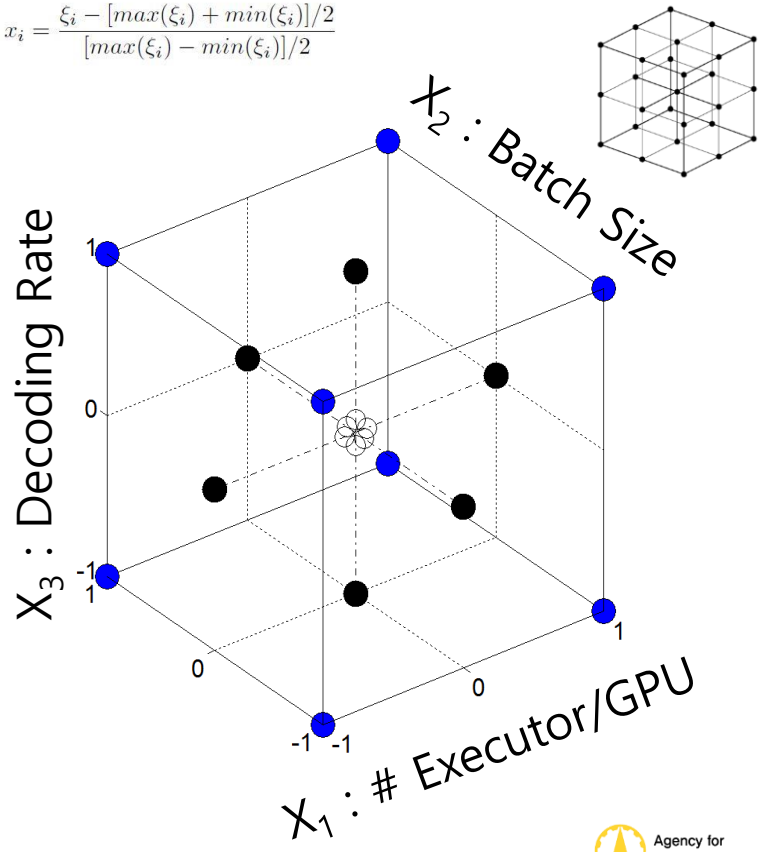
- DOE/RSM approach for system optimization
  - Factor levels for the FCCD experiments

Design	No.	Remark
Central Composite Design (FCCD)		
Total Runs	20	pts
Factorial	8	pts
Center	6	pts
Axial	6	pts
Factors	Factorial values	
X1 : Executer/GPU	1~3	
X2 : Batch Size	1~15	
X3 : Decoding Rate	1~9	Hz
Response		
Y1 : Throughput	1,800≤ Y1 ≤ 2,500	Images/sec
Y2 : Processing Time	0.5 ≤ Y2 ≤ 1.0	sec

			Natural level (ξ_i)			Coded level (x_i)		
Name	Factor	Units	Low	Center	High	Low	Center	High
Executor	X_1	-	1	2	3	-1	0	+1
Batch Size	X_2	-	1	8	15	-1	0	+1
Decoding Rate	X_3	Hz	1	5	9	-1	0	+1

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{ii} x_i^2 + \sum_{i=1}^{k-1} \sum_{j=2}^{k} \beta_{ij} x_i x_j + \epsilon \quad \text{for } n \text{ observations}$$

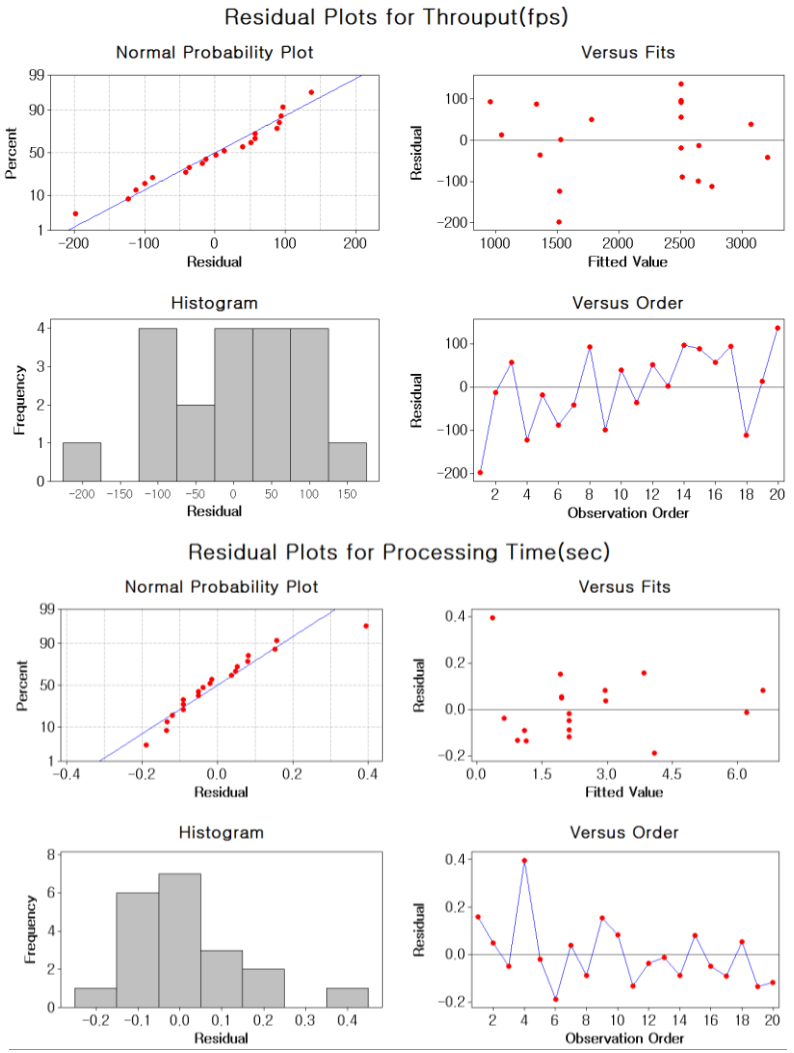
where $x_i = \dfrac{\xi_i - [max(\xi_i) + min(\xi_i)]/2}{[max(\xi_i) - min(\xi_i)]/2}$



Solution: the statistical DOE based system optimization

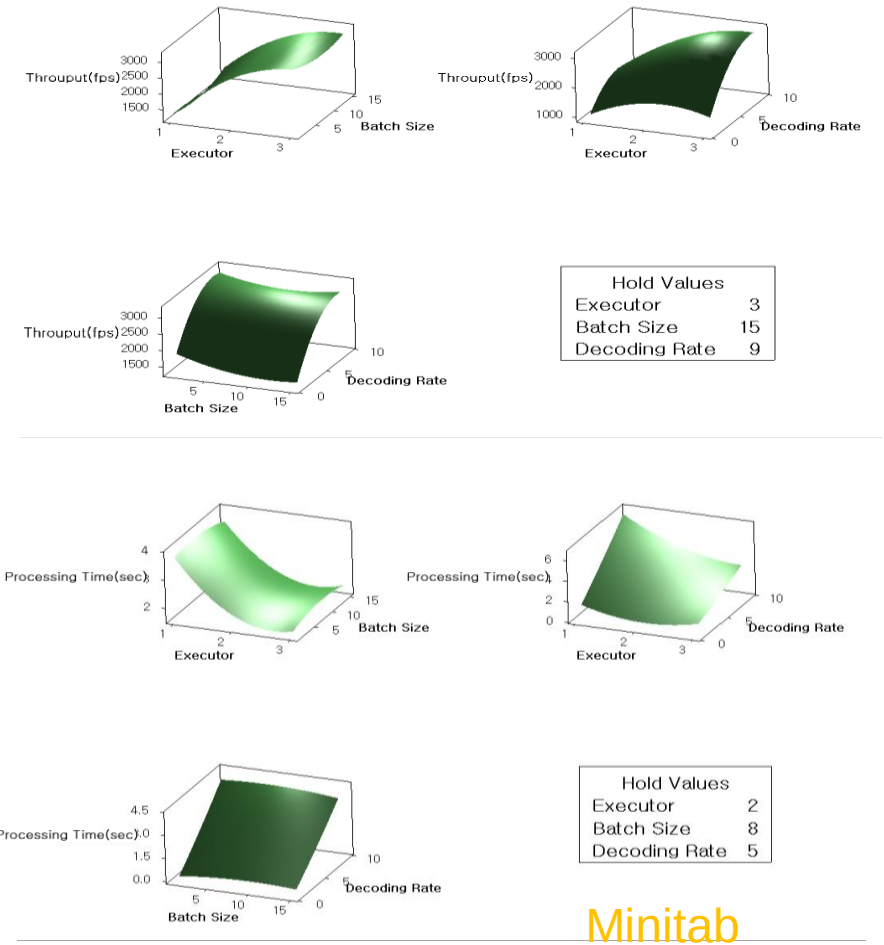
- FCCD matrix and measured results

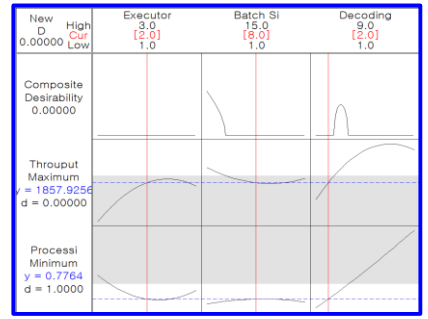
RUN No	Executor(x1) X1	Bach Size(x2) X2	Decoding Rate(x3) X3	Throughput(y1) [images/sec] Y1	Processing Time(y2) [sec] Y2
1	1	8	5	1315	4.00
2	3	8	5	2640	2.00
3	2	8	5	2565	2.07
4	2	8	1	1396	0.75
5	2	8	5	2490	2.10
6	2	8	9	2430	3.90
7	3	1	9	3168	3.00
8	2	8	5	2600	2.03
9	2	15	5	2545	2.07
10	3	15	9	3114	3.03
11	3	15	1	1325	0.80
12	3	1	1	1832	0.58
13	1	15	9	1530	6.20
14	2	8	5	2605	2.03
15	1	1	9	1422	6.67
16	2	8	5	2565	2.07
17	1	15	1	1052	1.00
18	2	1	5	2645	2.00
19	1	1	1	1062	1.00
20	2	8	5	2645	2.00

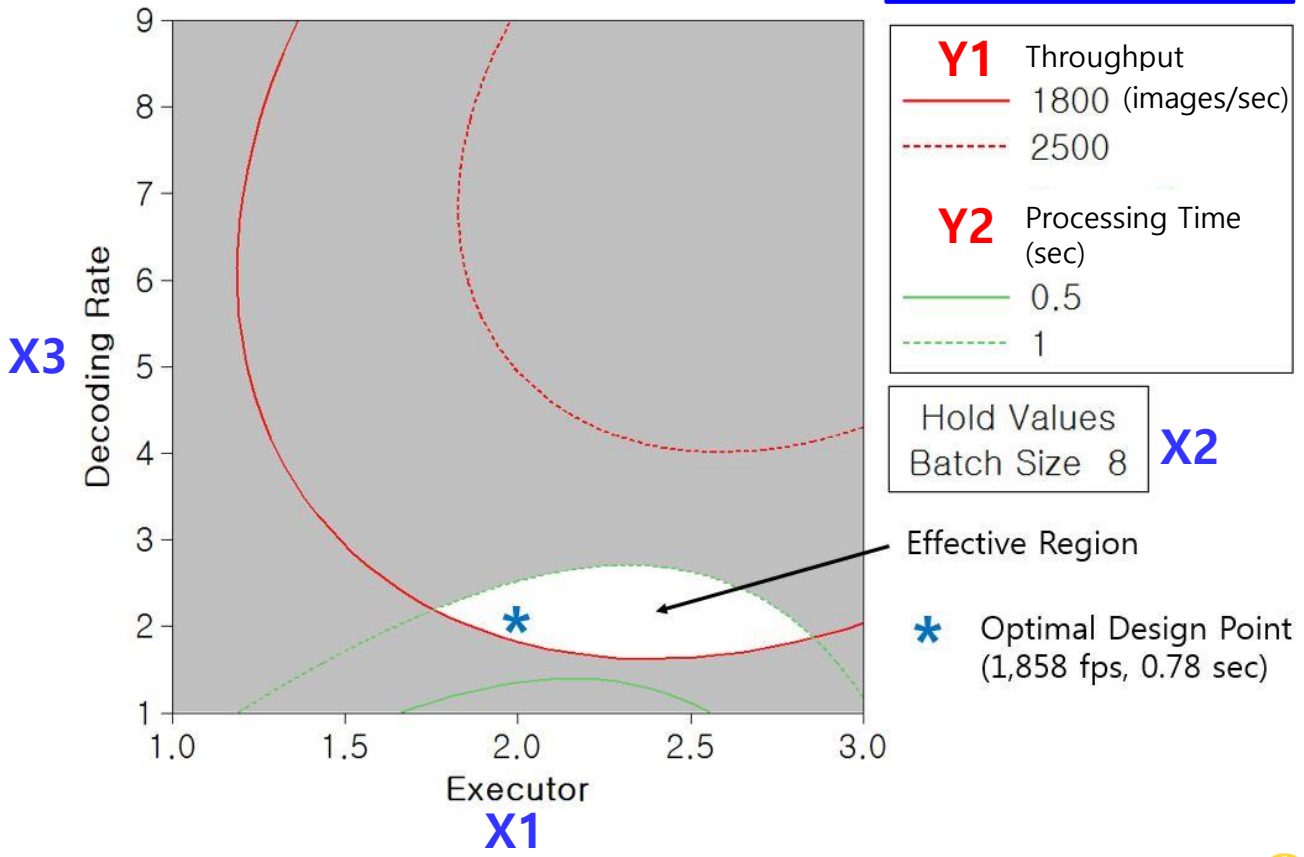


Solution: the statistical DOE based system optimization

- Optimization results: overlaid response surface plot



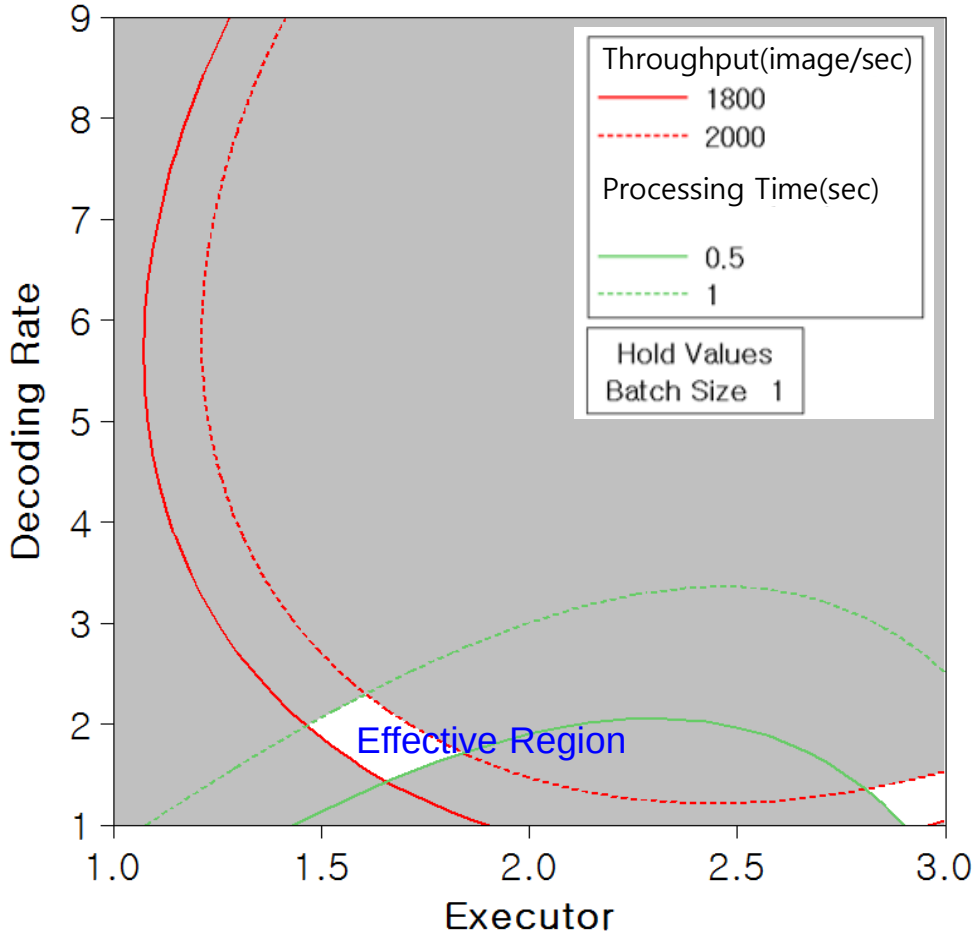




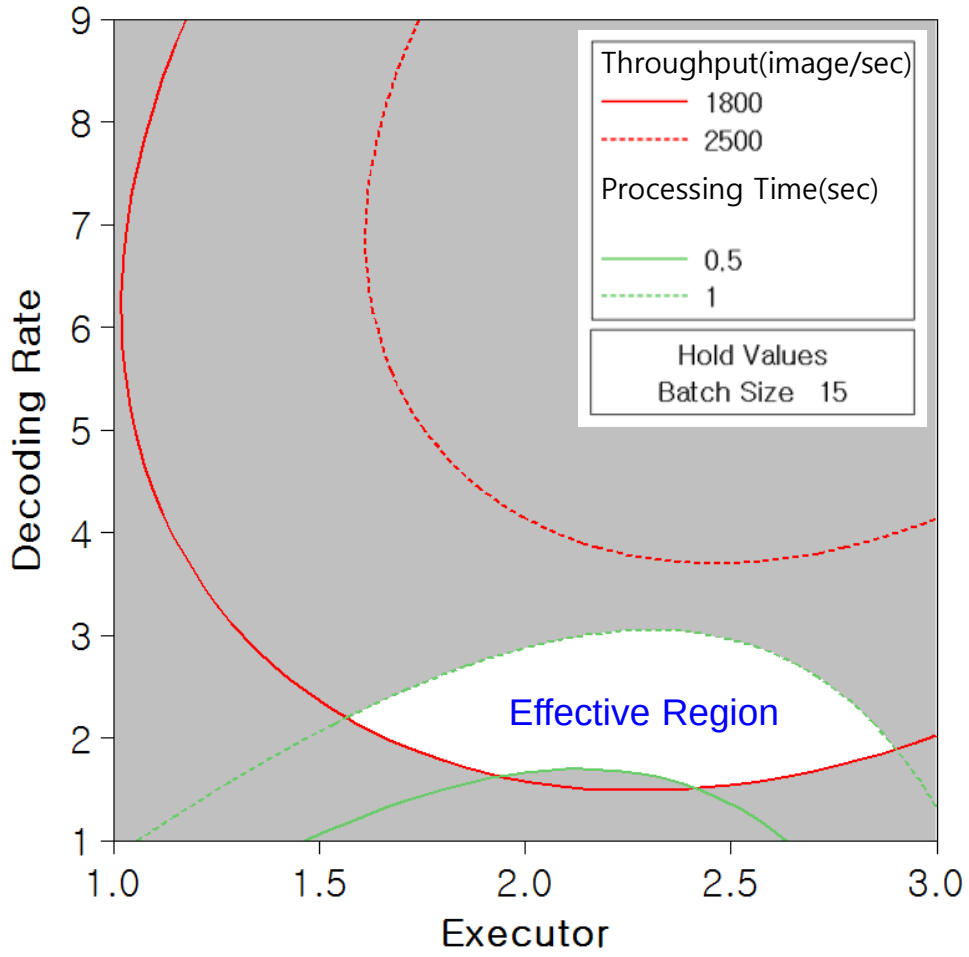
Solution: the statistical DOE based system optimization

- Optimization results for other design requirements

X3



X1



Part 2 - Summary (DL system optimization)

- We presented an efficient way to attain the optimal design parameters for distributed deep learning system with a statistical DOE method
- We applied the suggested approaches through various experiments according to the DOE/RSM based configurations.
- We demonstrated the effect of design parameters such as the number of Spark executors per GPU, batch size and video decoding rate for deep learning inference system in regards of performance metrics such as throughput and processing time.

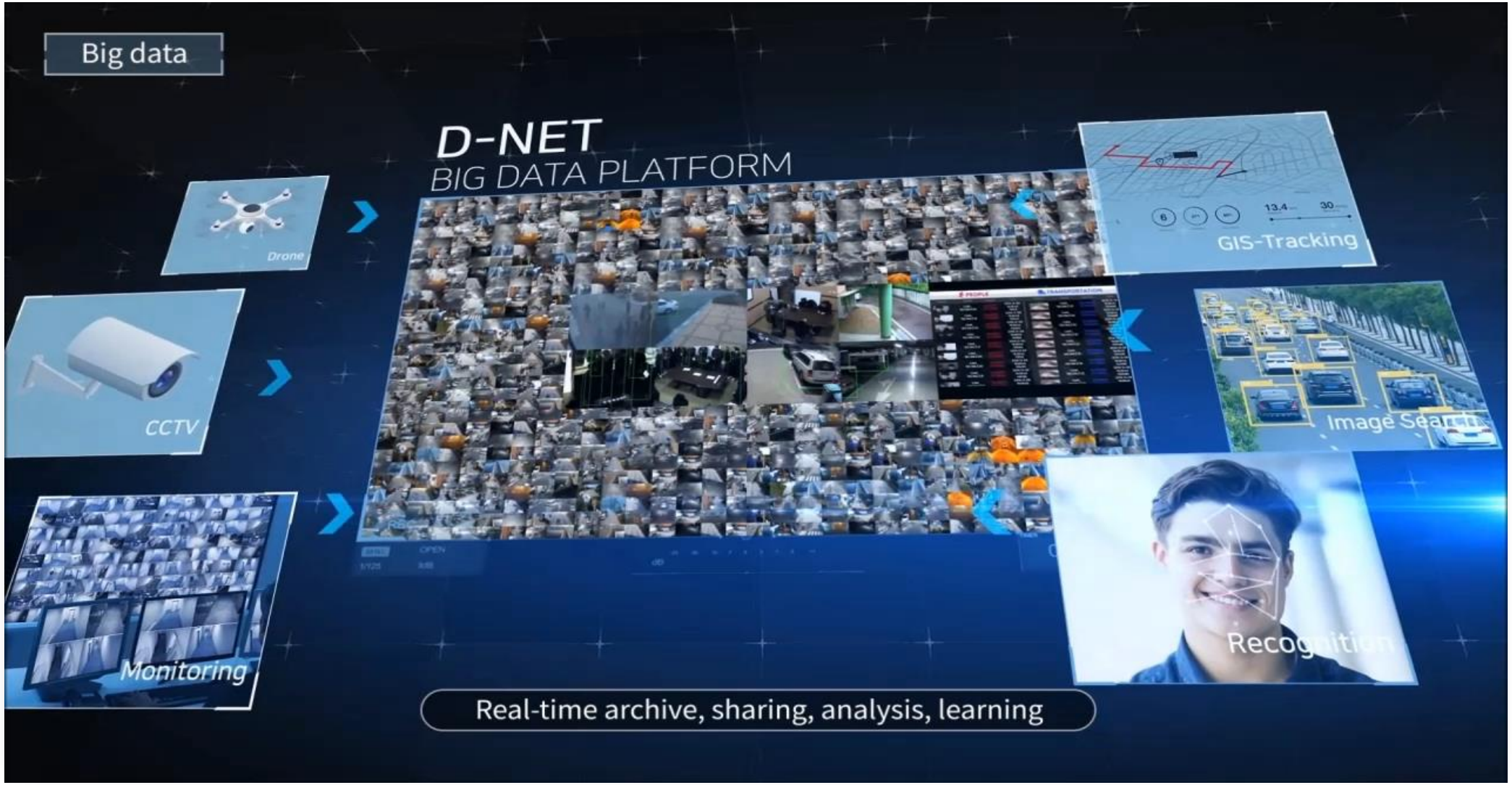
Outline

- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions

Use case: Intelligent Video Analytics

D-NET BIG DATA PLATFORM

- ‘D-NET’: GPU-accelerated big data analytics platform

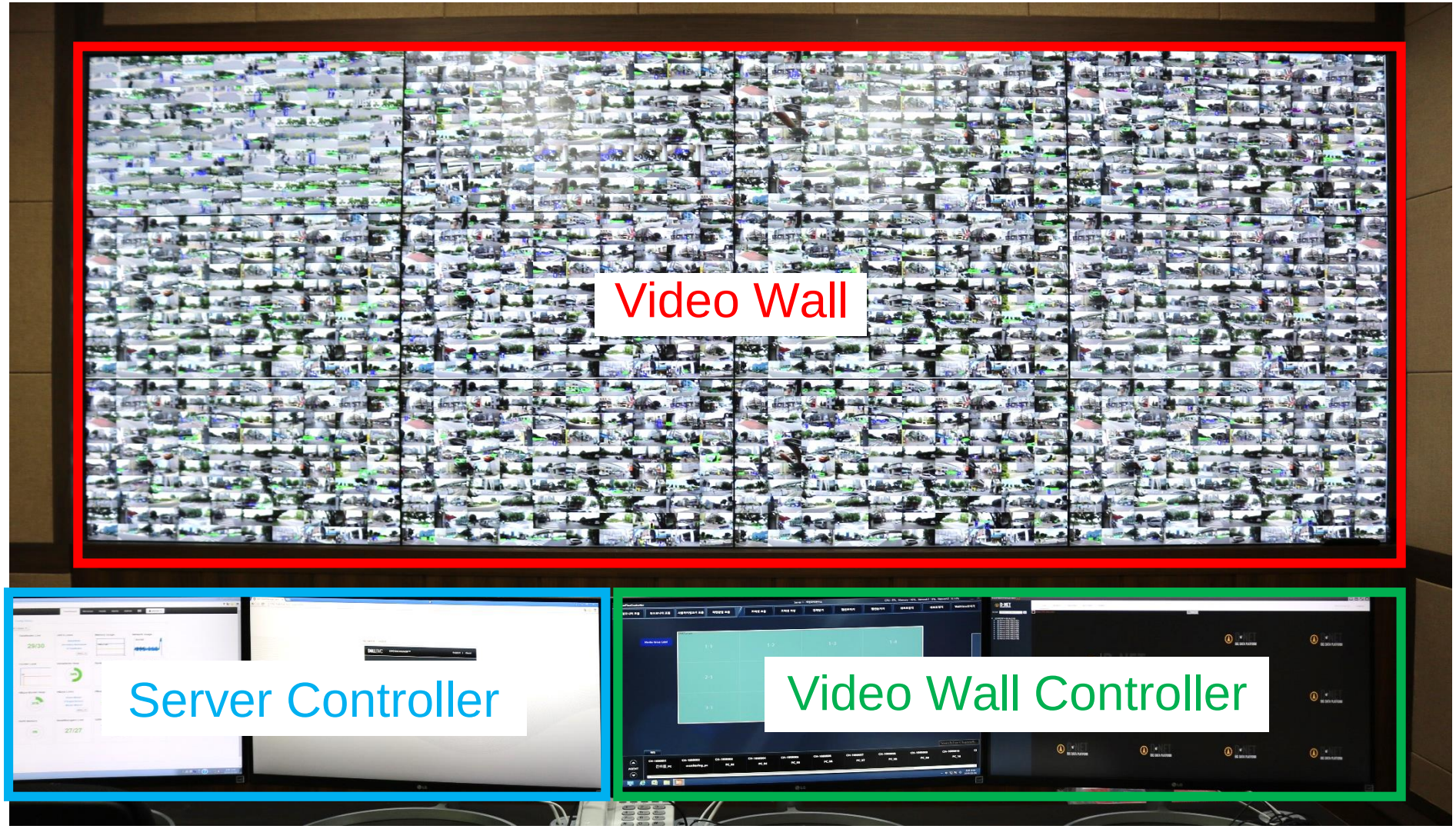


Use case: Intelligent Video Analytics

- Big data server room

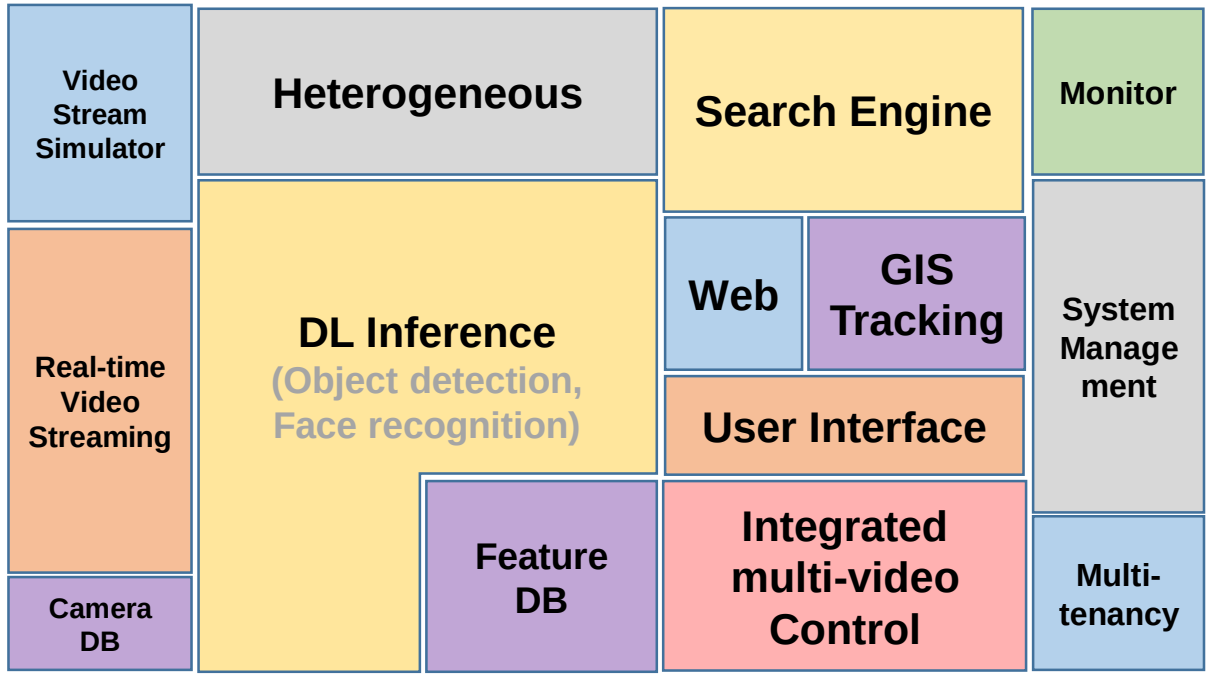
Use case: Intelligent Video Analytics

- Integrated control room



Use case: Intelligent Video Analytics

- Main functions
  - Video stream simulator
  - Real-time video streaming
  - DL-based object detection
  - Heterogeneous model deployment
  - GIS-based object tracking
  - Object tracking by face recognition
  - DB-based object search
  - Cluster resource monitoring
  - Integrated multi-video control

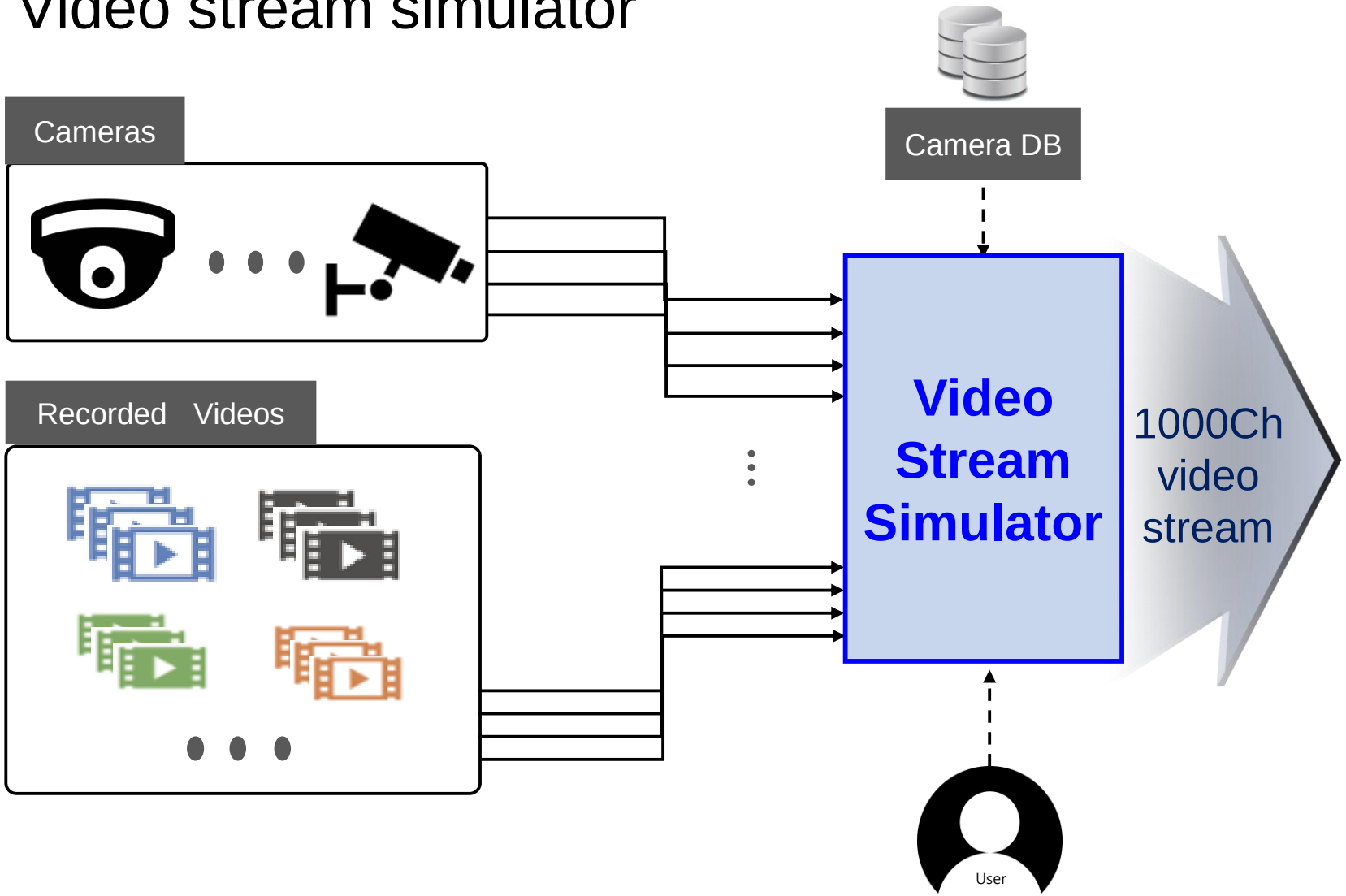


Real-time Intelligent Video Analytics for 1000Ch CCTV streams

NVIDIA Tesla M60 GPU-accelerated DL Inference on 31-node & 30-GPU Hadoop-Spark Cluster

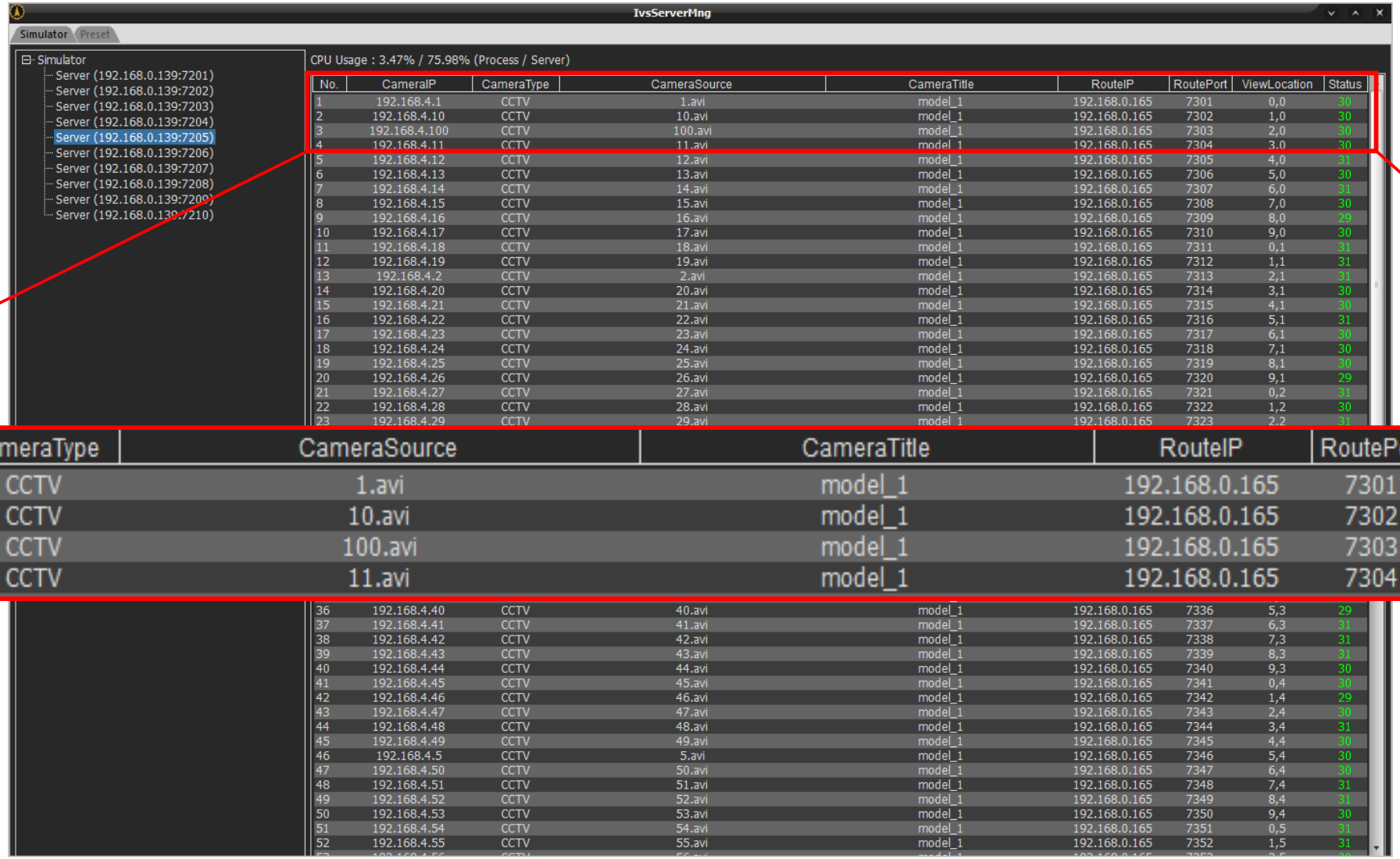
Use case: Intelligent Video Analytics

- Video stream simulator



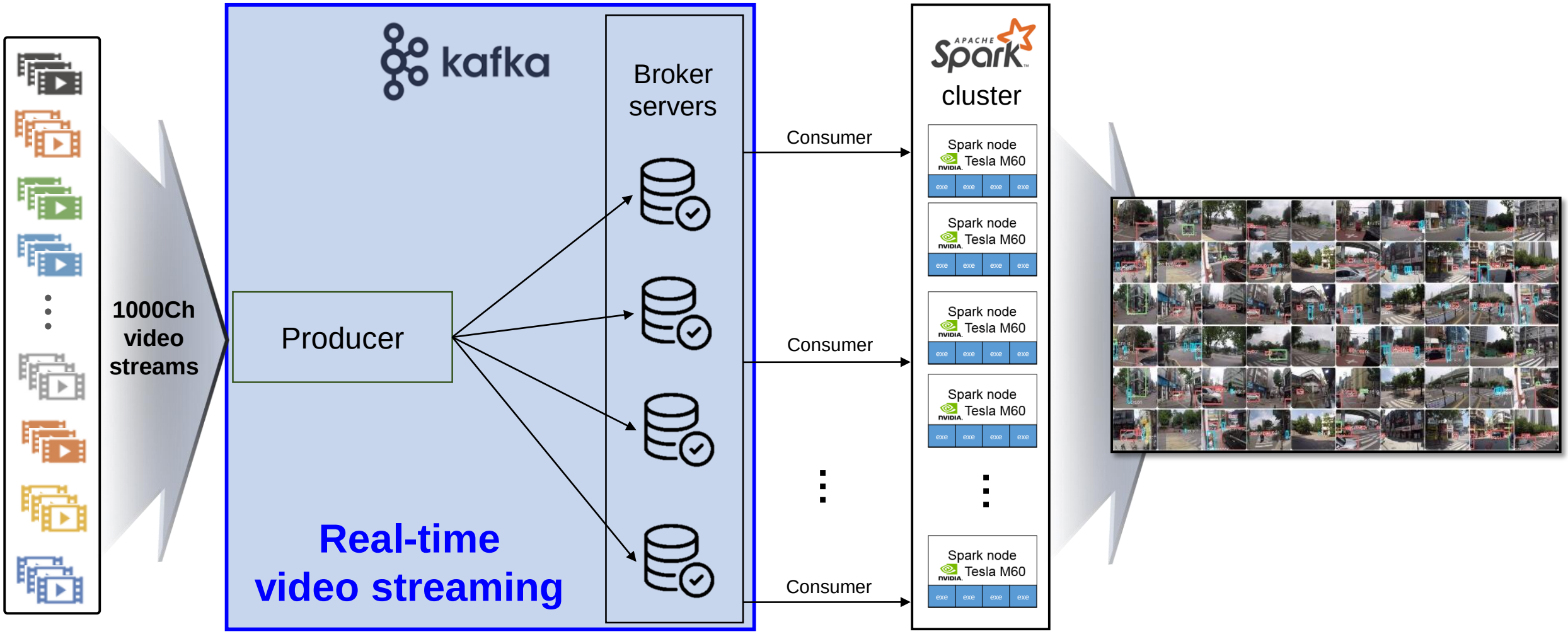
Use case: Intelligent Video Analytics

- Video stream simulator



Use case: Intelligent Video Analytics

- Real-time video streaming

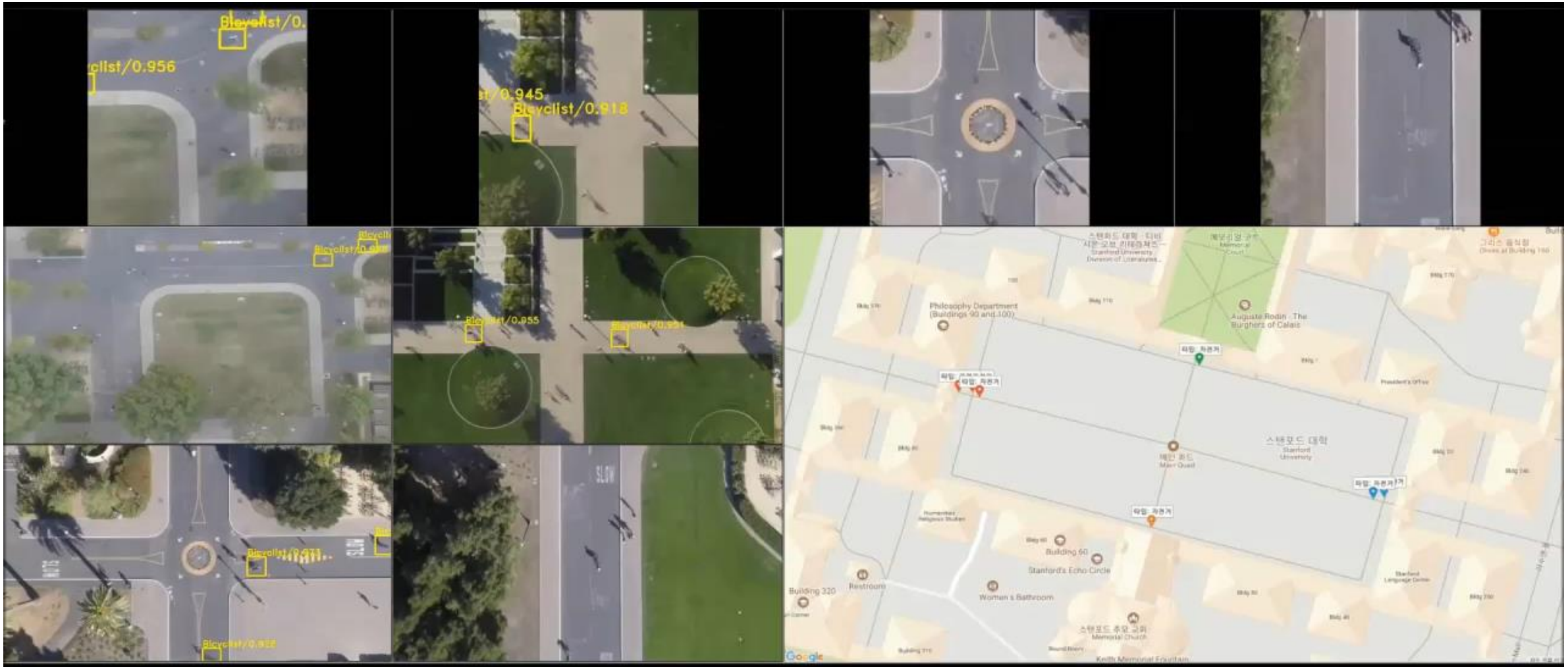


Use case: Intelligent Video Analytics

- Deep learning based object detection (CCTV)

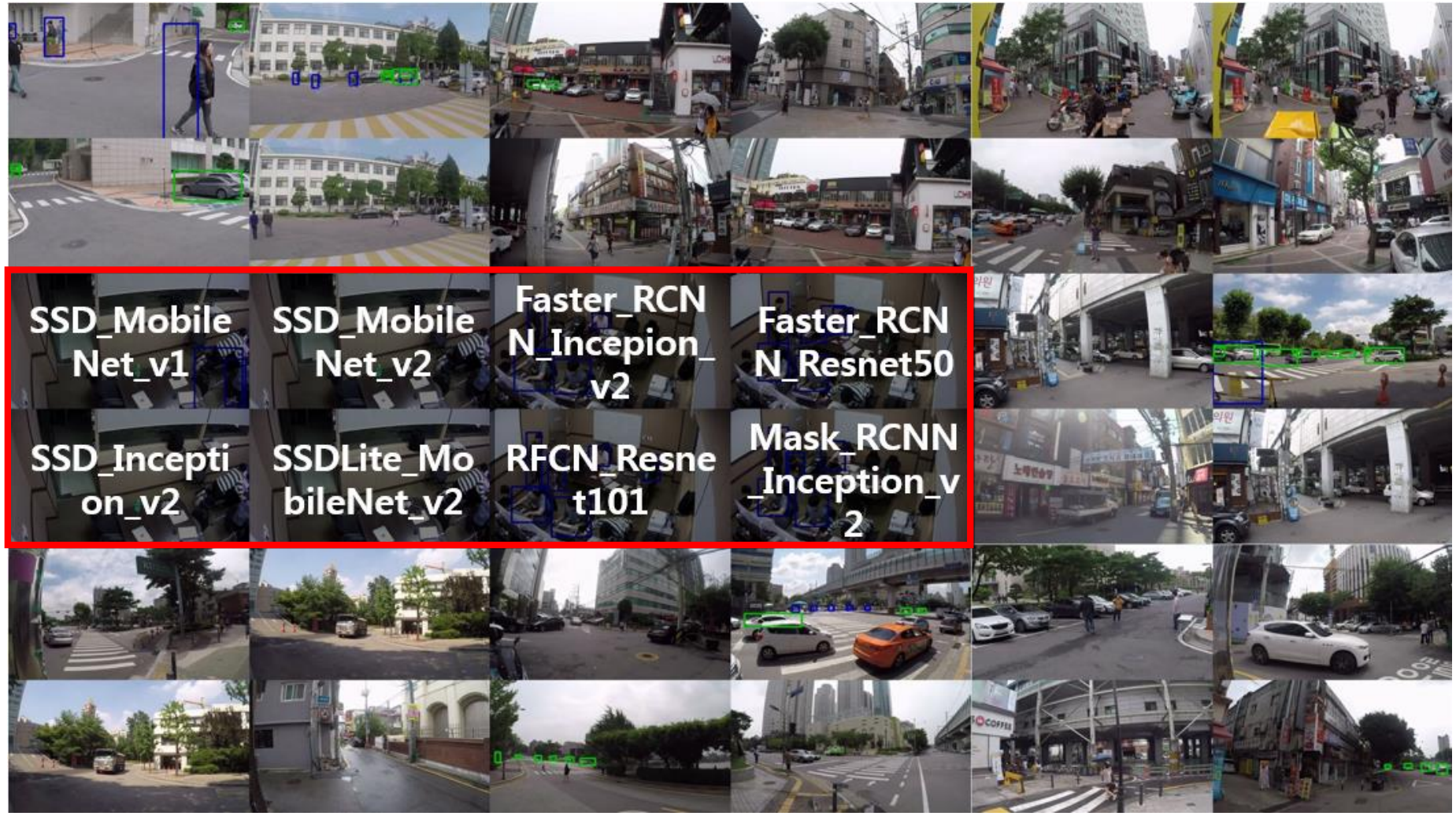
Use case: Intelligent Video Analytics

- Deep learning based object detection (aerial view)



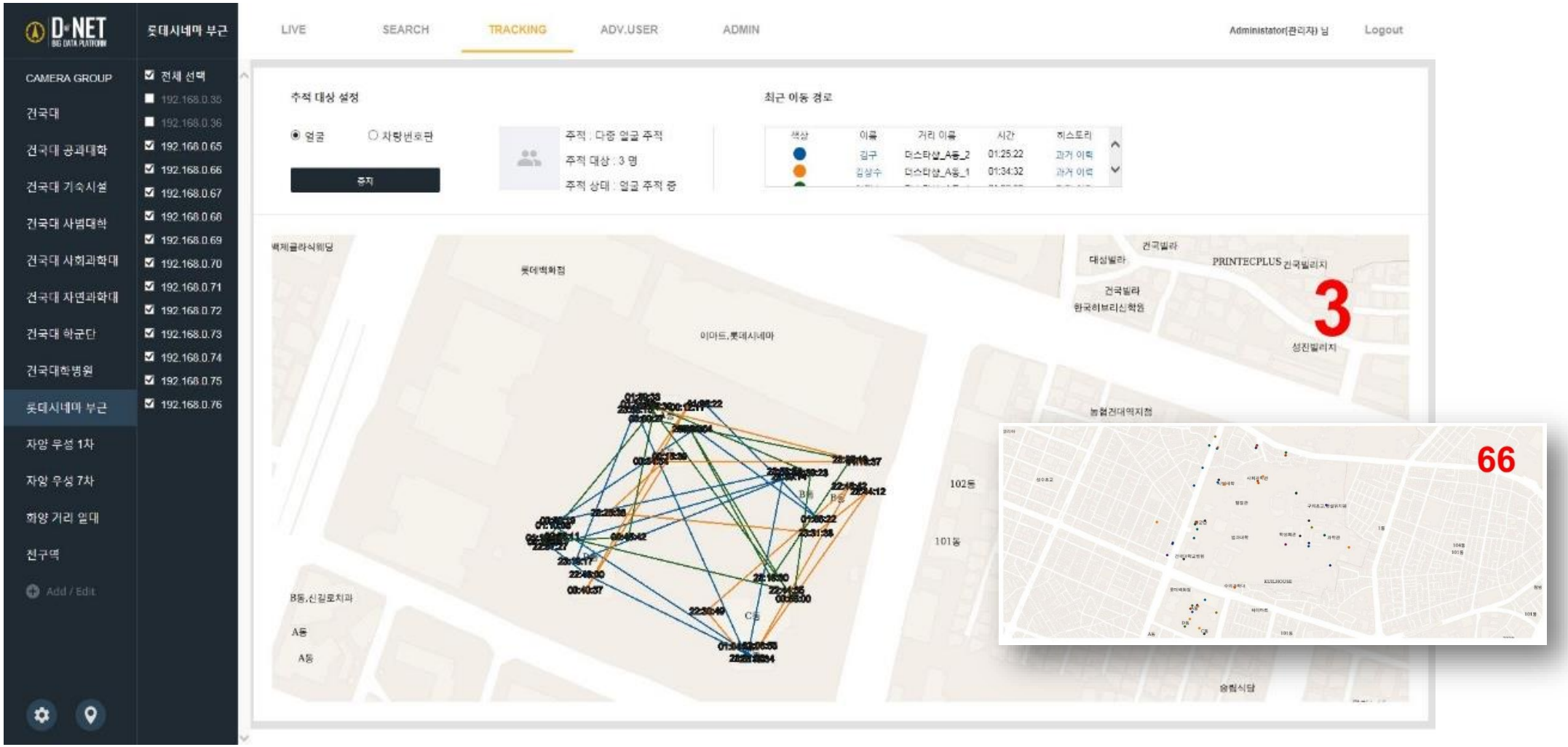
Use case: Intelligent Video Analytics

- Heterogeneous model deployment



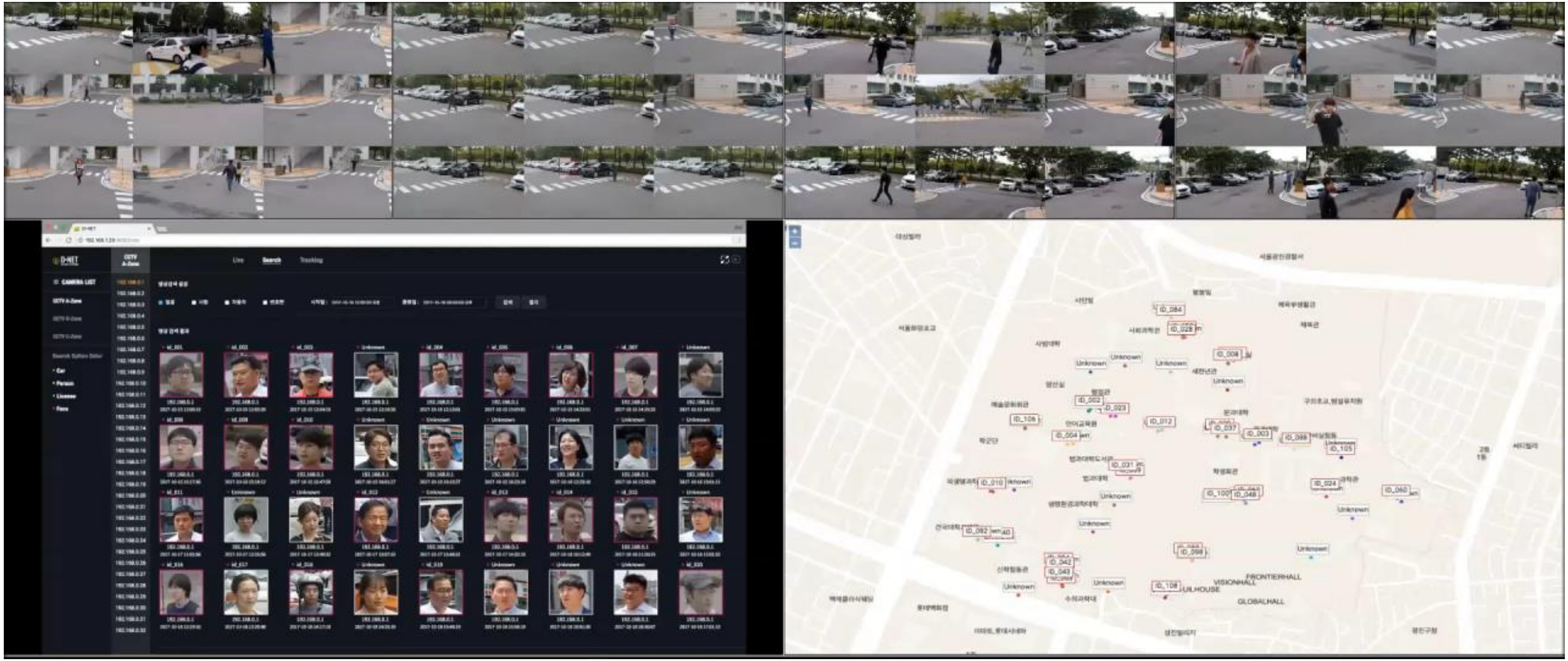
Use case: Intelligent Video Analytics

- GIS-based object tracking



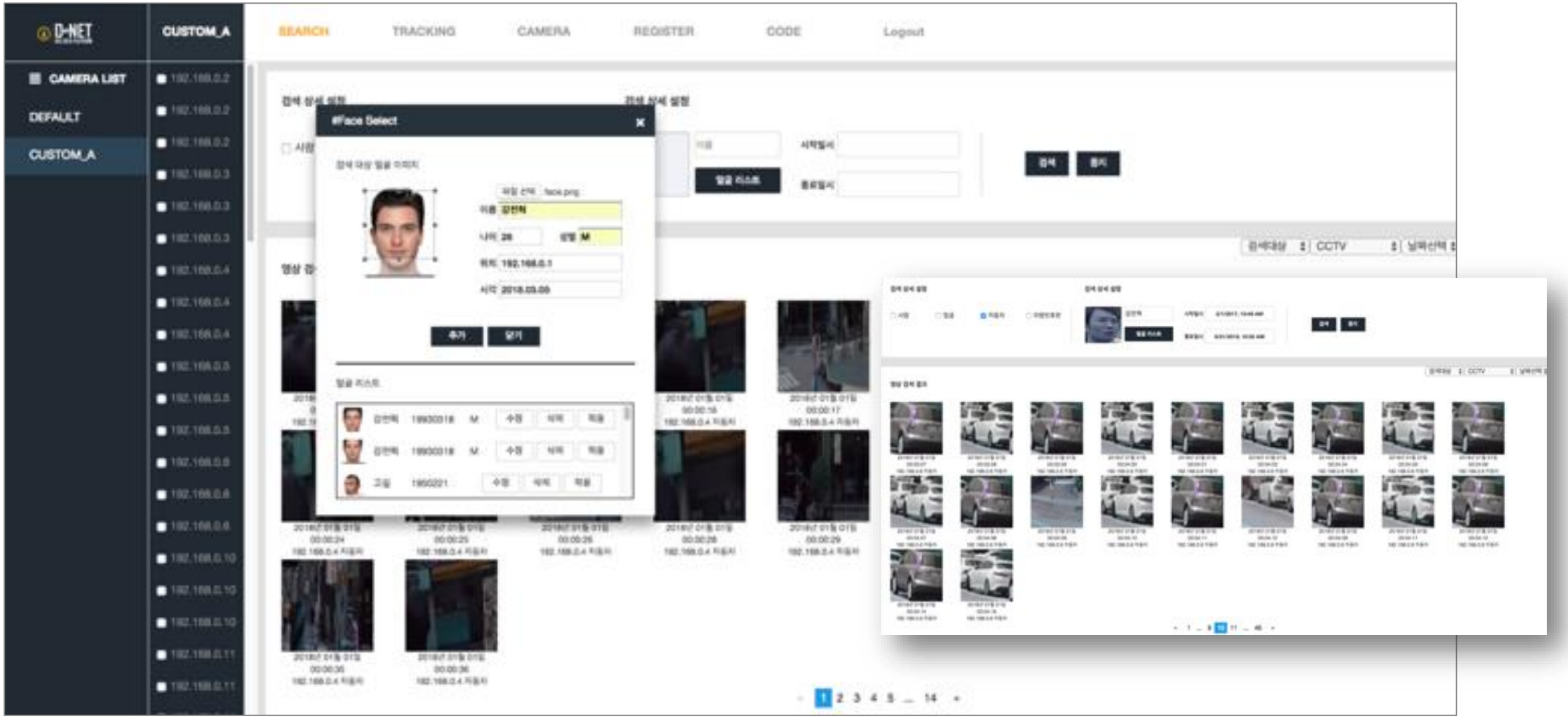
Use case: Intelligent Video Analytics

- Object tracking by face recognition



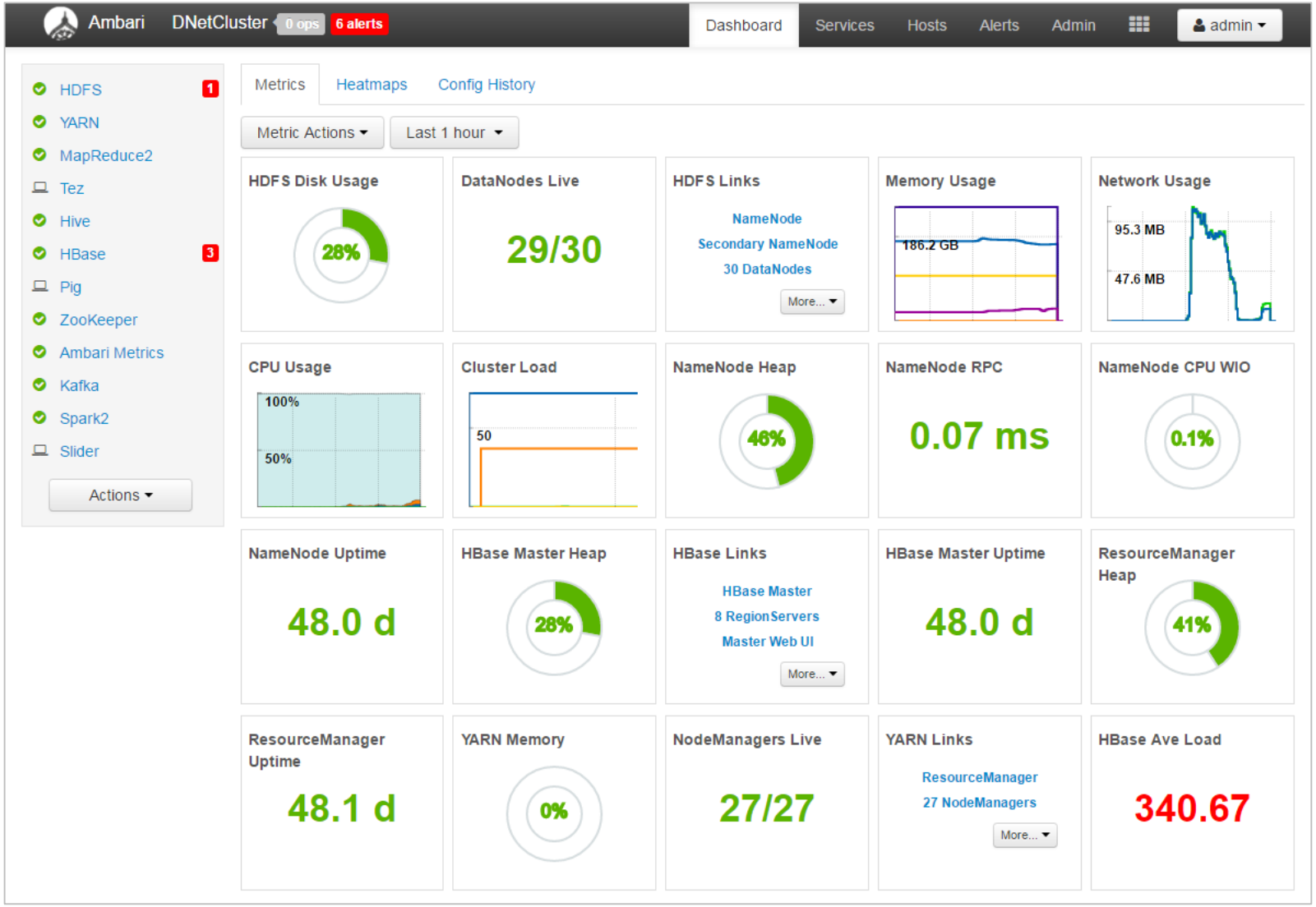
Use case: Intelligent Video Analytics

- Database based object search for cropped image



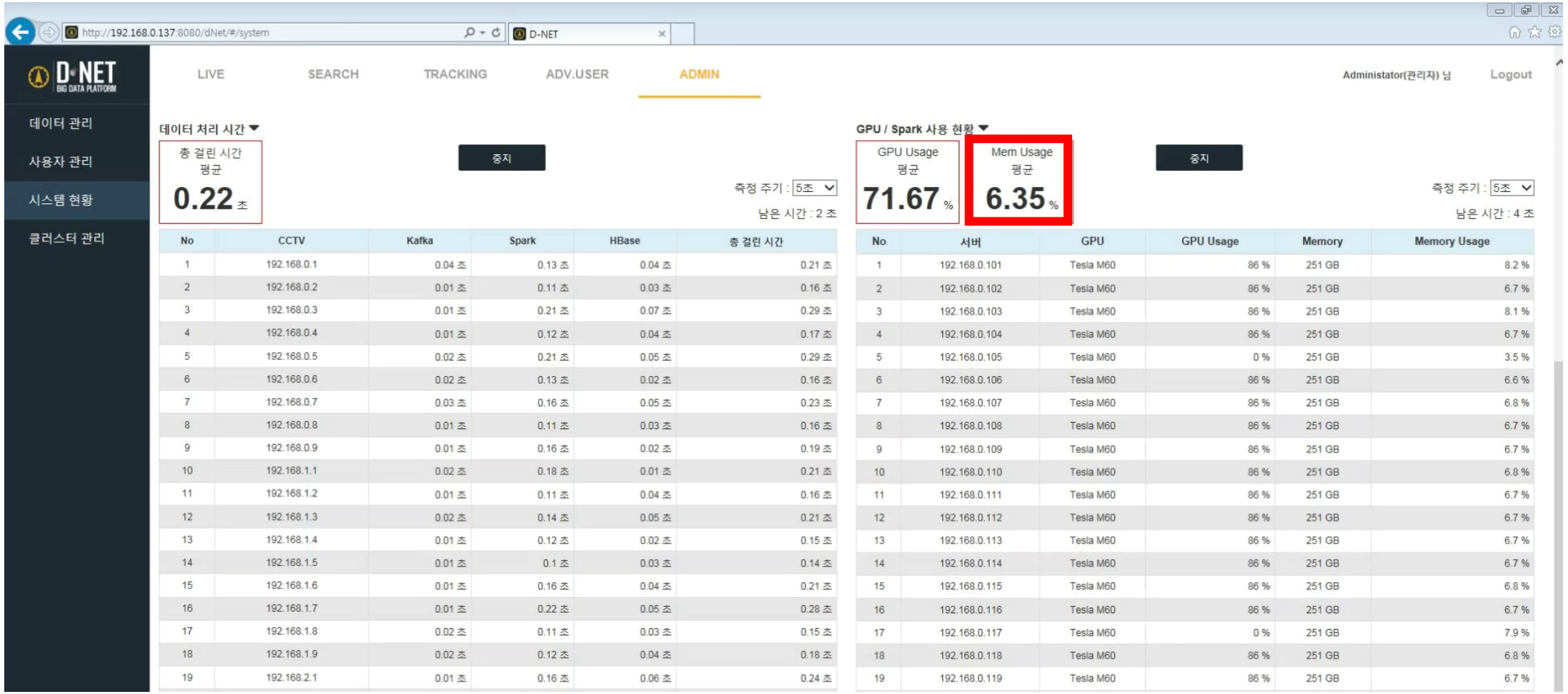
Use case: Intelligent Video Analytics

- Cluster resource monitoring



Use case: Intelligent Video Analytics

- Cluster resource monitoring

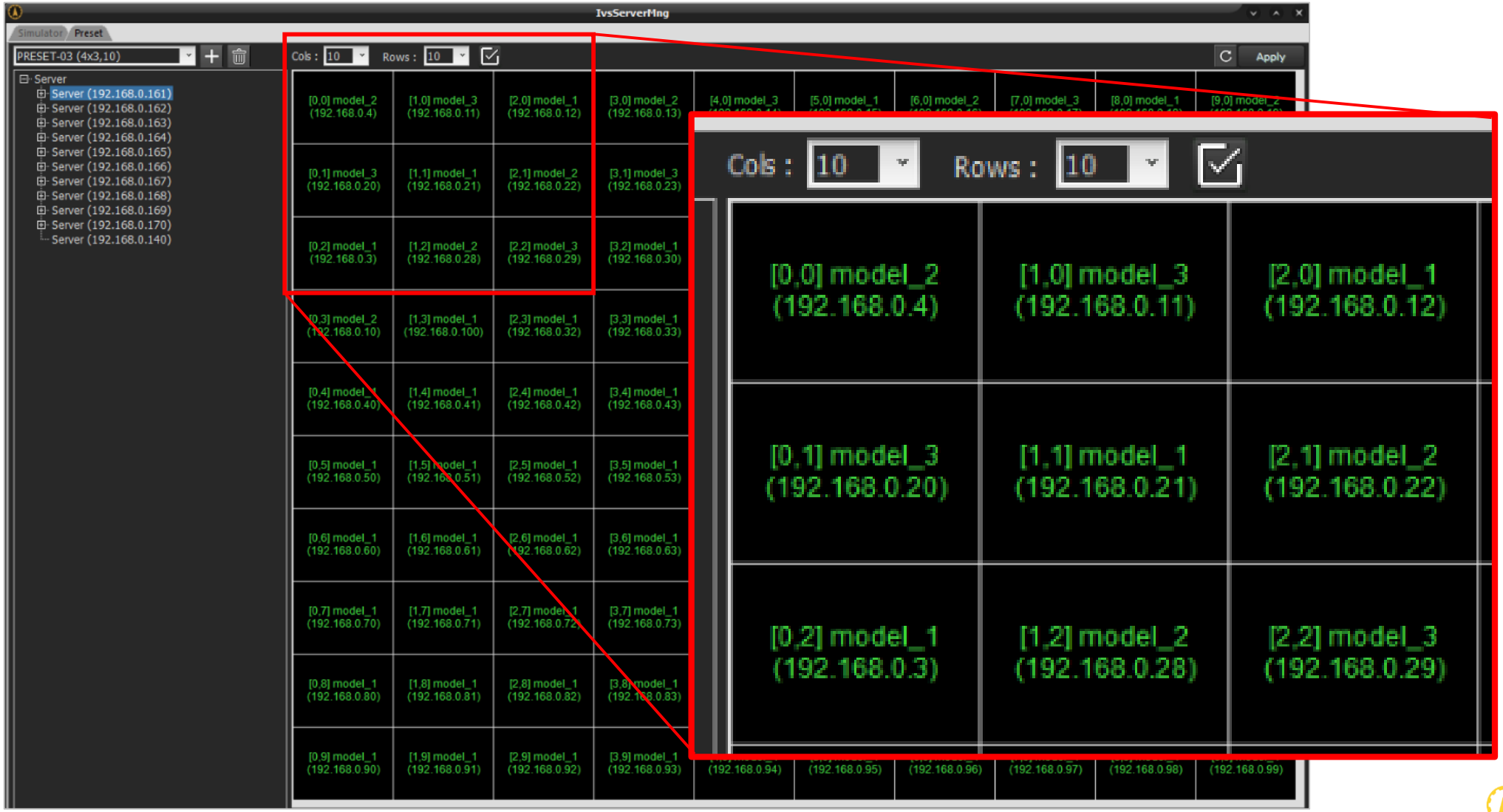


No	CCTV	Kafka	Spark	HBase	총 걸린 시간
1	192.168.0.1	0.04 초	0.13 초	0.04 초	0.21 초
2	192.168.0.2	0.01 초	0.11 초	0.03 초	0.16 초
3	192.168.0.3	0.01 초	0.21 초	0.07 초	0.29 초
4	192.168.0.4	0.01 초	0.12 초	0.04 초	0.17 초
5	192.168.0.5	0.02 초	0.21 초	0.05 초	0.29 초
6	192.168.0.6	0.02 초	0.13 초	0.02 초	0.16 초
7	192.168.0.7	0.03 초	0.16 초	0.05 초	0.23 초
8	192.168.0.8	0.01 초	0.11 초	0.03 초	0.16 초
9	192.168.0.9	0.01 초	0.16 초	0.02 초	0.19 초
10	192.168.1.1	0.02 초	0.18 초	0.01 초	0.21 초
11	192.168.1.2	0.01 초	0.11 초	0.04 초	0.16 초
12	192.168.1.3	0.02 초	0.14 초	0.05 초	0.21 초
13	192.168.1.4	0.01 초	0.12 초	0.02 초	0.15 초
14	192.168.1.5	0.01 초	0.1 초	0.03 초	0.14 초
15	192.168.1.6	0.01 초	0.16 초	0.04 초	0.21 초
16	192.168.1.7	0.01 초	0.22 초	0.05 초	0.28 초
17	192.168.1.8	0.02 초	0.11 초	0.03 초	0.15 초
18	192.168.1.9	0.02 초	0.12 초	0.04 초	0.18 초
19	192.168.2.1	0.01 초	0.16 초	0.06 초	0.24 초

No	서버	GPU	GPU Usage	Memory	Memory Usage
1	192.168.0.101	Tesla M60	86 %	251 GB	8.2 %
2	192.168.0.102	Tesla M60	86 %	251 GB	6.7 %
3	192.168.0.103	Tesla M60	86 %	251 GB	8.1 %
4	192.168.0.104	Tesla M60	86 %	251 GB	6.7 %
5	192.168.0.105	Tesla M60	0 %	251 GB	3.5 %
6	192.168.0.106	Tesla M60	86 %	251 GB	6.6 %
7	192.168.0.107	Tesla M60	86 %	251 GB	6.8 %
8	192.168.0.108	Tesla M60	86 %	251 GB	6.7 %
9	192.168.0.109	Tesla M60	86 %	251 GB	6.7 %
10	192.168.0.110	Tesla M60	86 %	251 GB	6.8 %
11	192.168.0.111	Tesla M60	86 %	251 GB	6.7 %
12	192.168.0.112	Tesla M60	86 %	251 GB	6.7 %
13	192.168.0.113	Tesla M60	86 %	251 GB	6.7 %
14	192.168.0.114	Tesla M60	86 %	251 GB	6.7 %
15	192.168.0.115	Tesla M60	86 %	251 GB	6.8 %
16	192.168.0.116	Tesla M60	86 %	251 GB	6.7 %
17	192.168.0.117	Tesla M60	0 %	251 GB	7.9 %
18	192.168.0.118	Tesla M60	86 %	251 GB	6.8 %
19	192.168.0.119	Tesla M60	86 %	251 GB	6.7 %

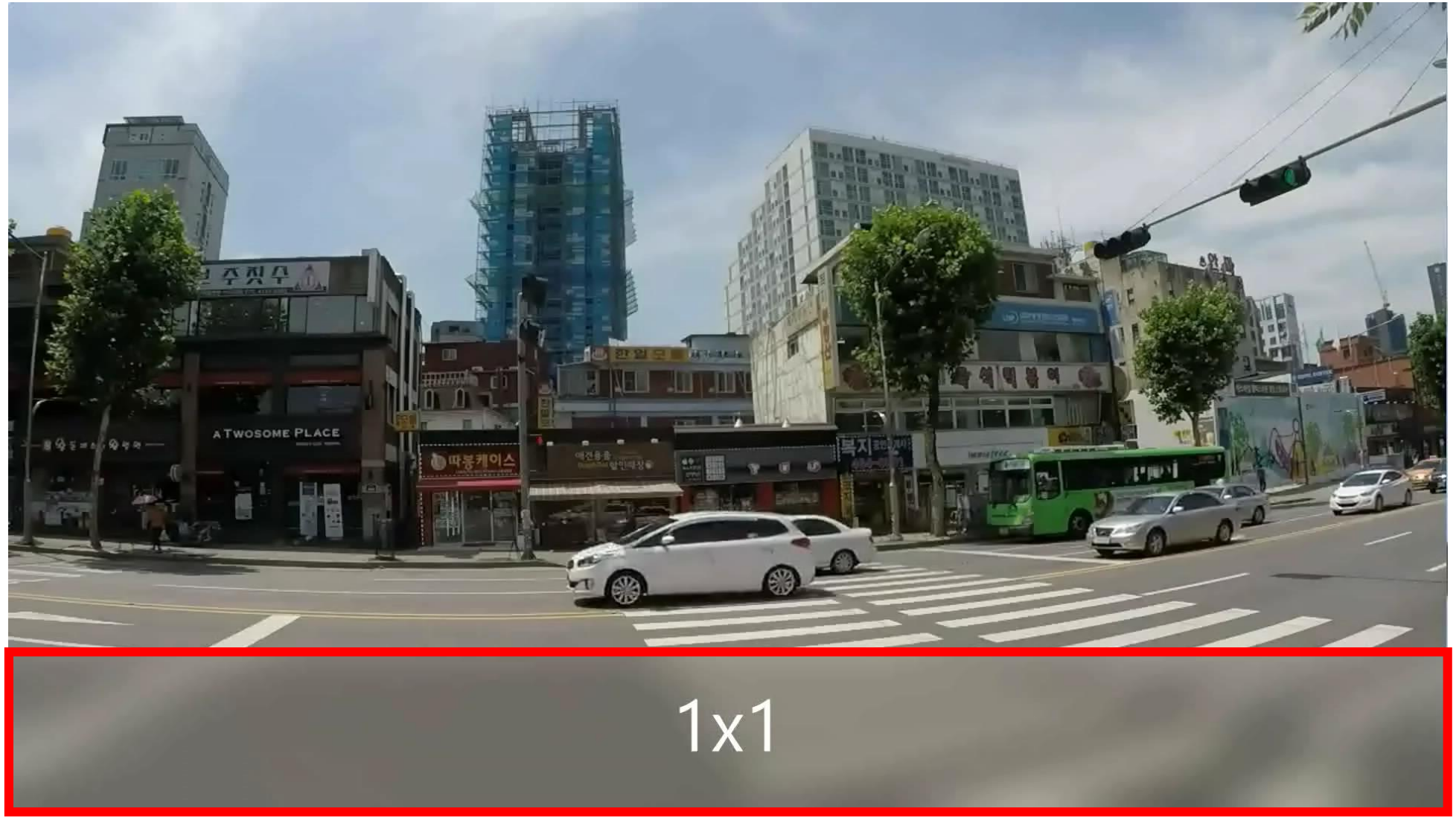
Use case: Intelligent Video Analytics

- Integrated multi-video control



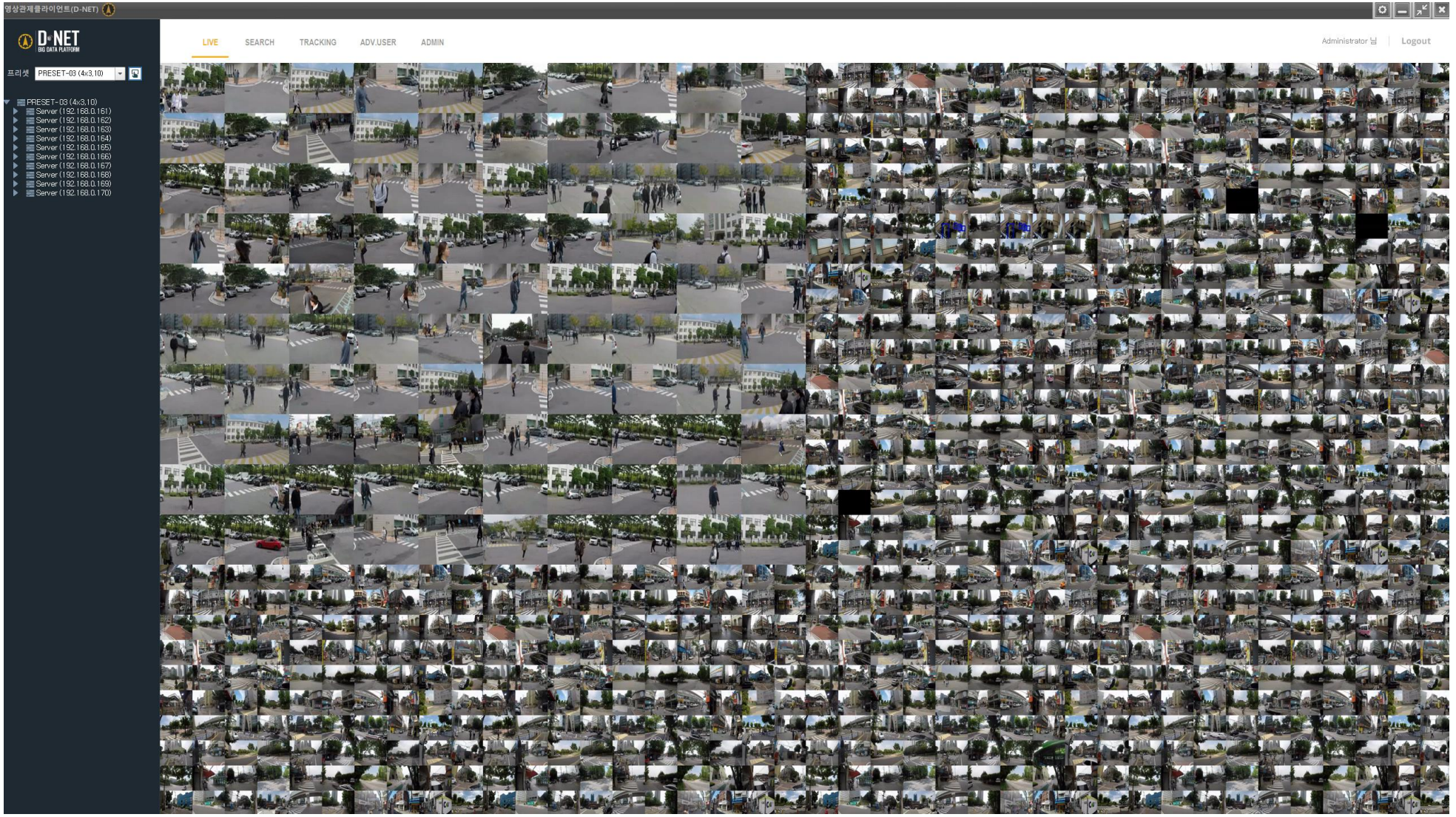
Use case: Intelligent Video Analytics

- Integrated multi-video control



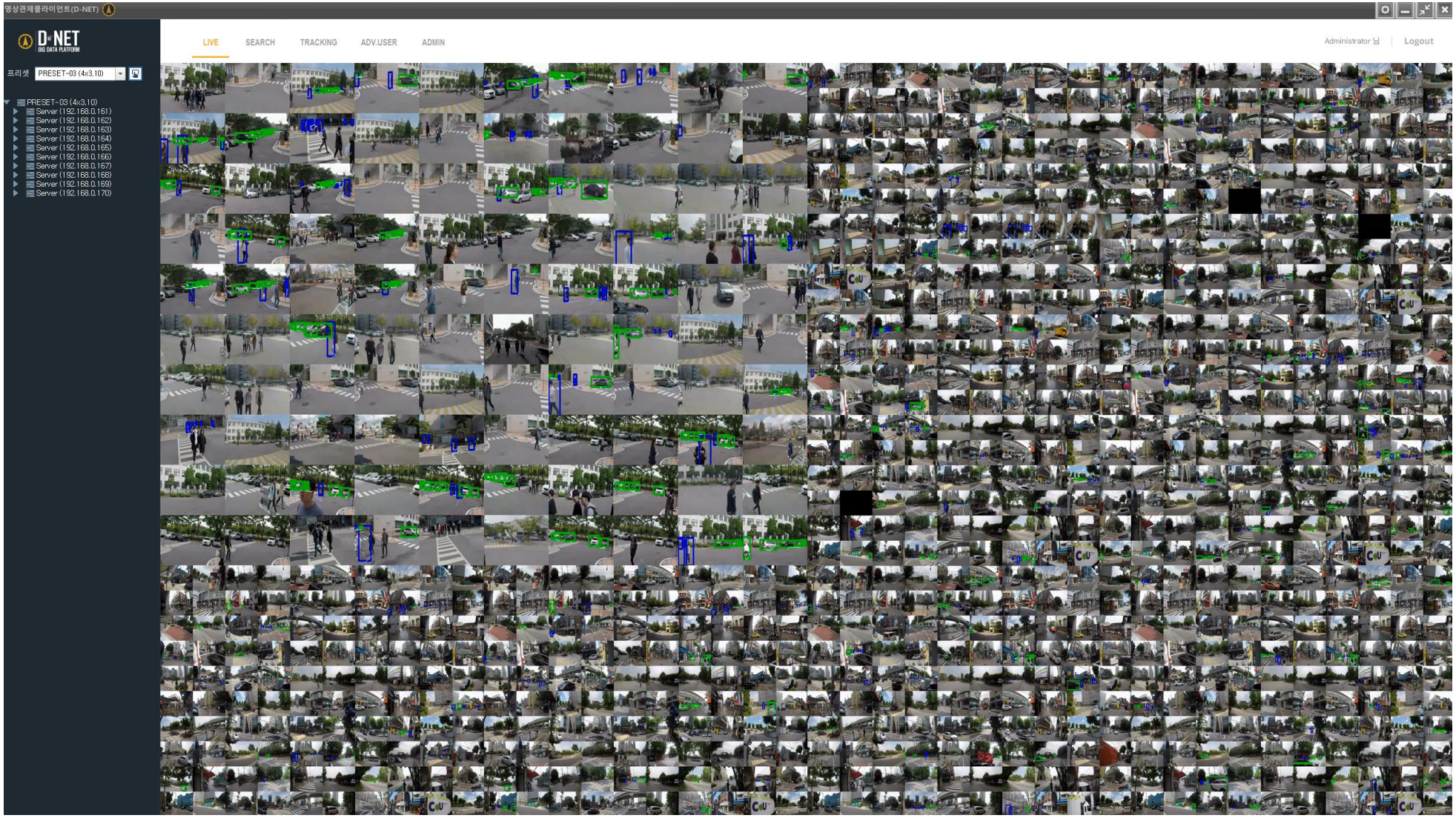
Use case: Intelligent Video Analytics

- Integrated multi-video control



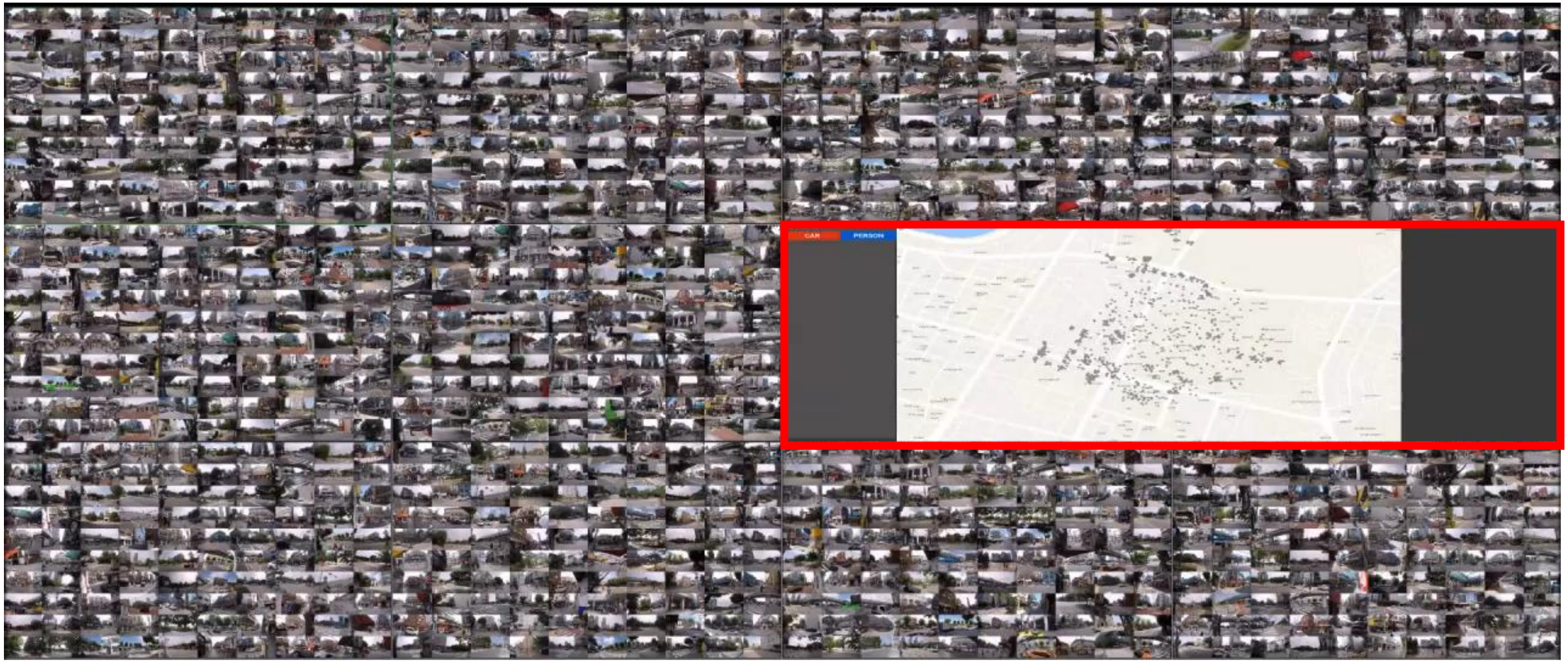
Use case: Intelligent Video Analytics

- Integrated multi-video control



Use case: Intelligent Video Analytics

- Large-scale Intelligent Video Analytics



Real-time Intelligent Video Analytics for 1000Ch CCTV streams

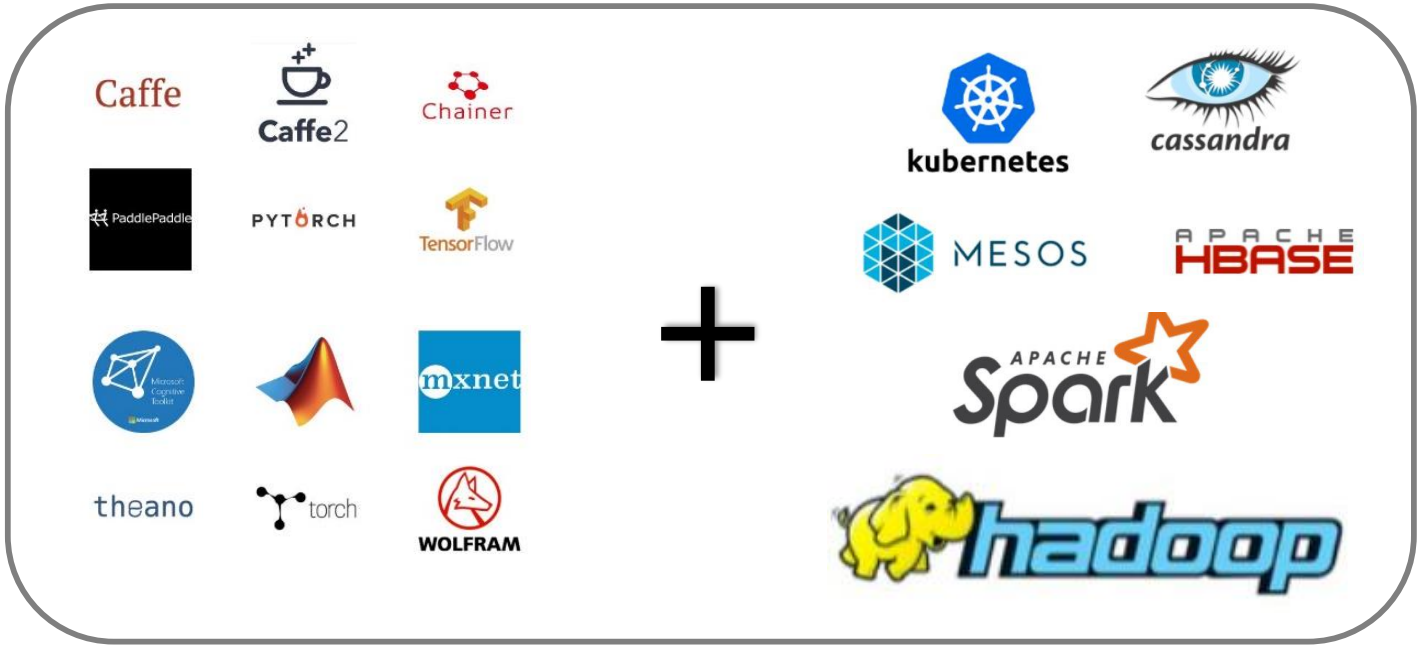
NVIDIA Tesla M60 GPU-accelerated DL Inference
on 31-node & 30-GPU Hadoop-Spark Cluster

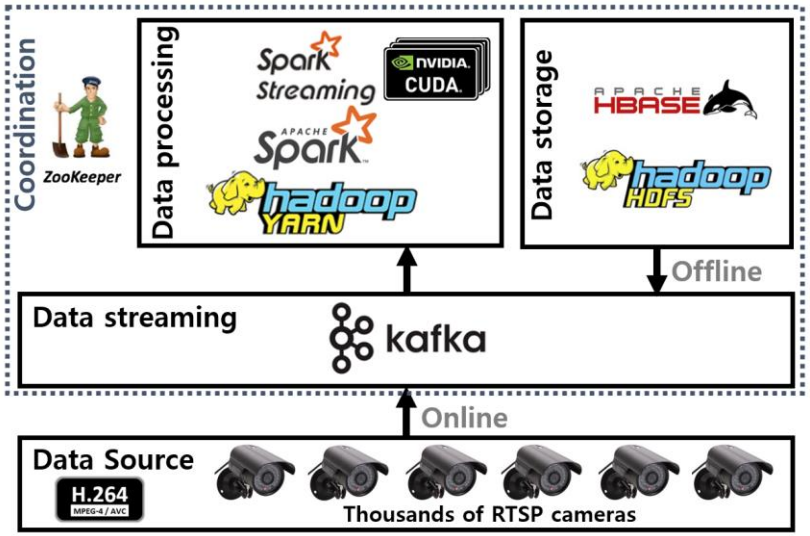
Outline

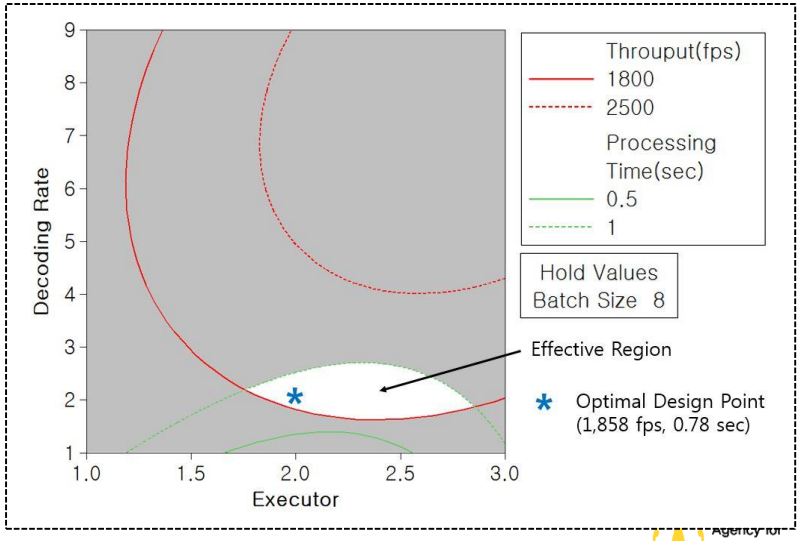
- Introduction
- GPU-accelerated DL Inference with Hadoop-Spark
  - Fast & Scalable DL Inference
  - DL System Optimization
  - Use case: Intelligent Video Analytics
- Conclusions

Conclusions

- Hadoop-Spark DL Inference works
- DOE gives significant efficiency gains
- Exciting IVA works more to come!







Thanks!

Hunmin Yang

hmyang@add.re.kr

Agency for Defense Development

Se-Yoon Oh

syoh@add.re.kr

Agency for Defense Development

Ki-Jung Ryu

taurus@add.re.kr

Agency for Defense Development

Acknowledgements

www.xiilab.com
Inception Startup

굿모닝아이텍(주) www.goodmit.co.kr