Computational Storage for Big-Data Analytics

Bala Samynathan
Co-Founder and Technical Director, Bigstream
Big Data Application Performance

- Machine Learning
- Natural Language Processing
- IoT/Edge Analytics
- Behavioral Analytics
- Risk Management
- Security Analytics
Hardware Accelerators Break Through the Processing Wall
Pushing Intelligence near Data

Today’s Architecture

- DRAM
- CPU
- PCIe
- SSD

- Heavy weight compute in CPU
- Large amount of data transfers
- TBs, PBs, EBs dataset

Performance Ceiling

- CPU performance bottleneck
- Large data movement
- IO bandwidth bottleneck
Pushing Intelligence near Data

Today’s Architecture

- Heavy weight compute in CPU
- Large amount of data transfers
- TBs, PBs, EBs dataset

Storage and Data Acceleration

- Minimal data movement
- Large internal bandwidth

Performance Ceiling

- CPU performance bottleneck
- Large data movement
- IO bandwidth bottleneck

Scale Performance with Data

- Concurrent processing on Big Data
- Near data processing
- Faster time to insight
Samsung SmartSSD Architecture Overview
Introducing: Bigstream Hyper-acceleration Layer

BIG DATA PLATFORMS

- presto
- kafka
- Spark
- MySQL
- TensorFlow
- elastic
- Hive

Dataflow Adaptation Layer

Dataflow Compiler

Hypervisor

Zero code change

Cross platform

Intelligent, automatic computation slicing

Hybrid acceleration

2X to 10X acceleration

Many-cores

GPU

FPGA

SmartSSD
Apache Spark

Client Application
- Big Data Platform APIs

Master Node
- Application Master
- Catalyst

Resource Management Messages

Cluster Management

Physical Plan
- Extended Query Optimization Strategies

Tasks

Executor Node
- Node Manager
- Executors
- Spark Task

Many-cores

Resource Manager
val df = sparkSession.read.schema(schema).json(jsonRDD)
df.filter("country is not null and price is not null")
  .filter(df("country").equalTo("USA"))
  .groupBy("action_id").agg(count("price"), min("price"), max("price")).show()
The Lifecycle of a Query

1. SQL Query
2. Cluster Query Plan
3. Node LST
4. Accelerator Templates
   - Generating Control Plane
   - Generating Data Plane
5. Bigstream Dataflow Compiler
6. Bigstream Dataflow Runtime/Execute Stages
   - CPU
   - GPU
   - FPGA
Physical Plan For This Query

== Physical Plan ==

*HashAggregate(keys=[action_id], functions=[count(1), min(price), max(price)])

+- Exchange hashpartitioning(action_id, 200)

    +- *HashAggregate(keys=[action_id],
                     functions=[partial_count(1), partial_min(price), partial_max(price)])

    +- *Project [action_id, price]

    +- *Filter (country = 30)

    +- Scan ExistingRDD[timestamp, action_id, user_id, country, price]
Linear Stage Trace

Linear Stage Traces (LST)

- Scan
- Filter
- Project
- Partial HashAgg

Cluster Query Plan

Node LST

Accelerator Templates

Control

Data

C

G

F
Slicing & Mapping

Storage

Accelerator

CPU

NIC

Cluster Query Plan

Node LST

Accelerator Templates

Control

Data

C G F

Accelerator Templates
Generating the Control and Data Plane Code
SmartSSD Ecosystem

bigstream HYPER-ACCELERATION

Database of Templates
- Template 1
- Template 2
- Template 3

Database of IPs
- Engine 1
- Engine 2
- Engine 3

User Space

OS

NVMe / PCIe Driver

Hardware

Host CPU ↔ Host DRAM

Host Interface Controller

SSD ↔ FPGA

FPGA+SSD

OSS/Third Party
- Samsung
- Bigstream
- Xilinx

FPGA DRAM
1. Load FPGA bitstream based on acceleration template
1. Load FPGA bitstream based on acceleration template

2. Software configuration of FPGA to customize hardware template for the app
1. Load FPGA bitstream based on acceleration template
2. Software configuration of FPGA to customize hardware template for the app
3. Issue “accelerated” I/O + compute to pump data toward FPGA
1. Load FPGA bitstream based on acceleration template
2. Software configuration of FPGA to customize hardware template for the app
3. Issue "accelerated" I/O + compute to pump data toward FPGA
4. FPGA copies the result to the application user space
1. Load FPGA bitstream based on acceleration template
2. Software configuration of FPGA to customize hardware template for the app
3. Issue “accelerated” I/O + compute to pump data toward FPGA
4. FPGA copies the result to the application user space
TPC-DS Scan Heavy Query - Cluster Results

~4x on average
~6x scan heavy

Query Number

<table>
<thead>
<tr>
<th>Query Number</th>
<th>Vanilla Spark Runtime (s)</th>
<th>Bigstream Spark Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>63</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>73</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>82</td>
</tr>
<tr>
<td>7</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>8</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>71</td>
<td>71</td>
</tr>
</tbody>
</table>

Processor: Intel Xeon Gold 6152 : 22 Cores with one SmartSSD per node. Memory: 200G per node
Spark Config: One master 4 executor nodes. 6 spark cores per node.
CPU Utilization Measurements

![Graph showing CPU utilization measurements for Spark and Bigstream over time.](image-url)
Bigstream FPGA Templates

Shell (Provided by FPGA Vendor)

Bigstream Shim - Adaptation Layer

Bigstream Reconfigurable Core

Row/Column Parser → SQL PushDown
Database Templates RoadMap

Bigstream SW Stack

<table>
<thead>
<tr>
<th>Scan Engines</th>
<th>SQL PushDown Engines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression (5 var.)</td>
<td>Project/Filter (40 var.)</td>
</tr>
<tr>
<td>JSON</td>
<td>Sort</td>
</tr>
<tr>
<td>CSV</td>
<td>Hashaggregate/</td>
</tr>
<tr>
<td>AVRO*</td>
<td>Sortaggregate (16 var.)</td>
</tr>
<tr>
<td>Parquet</td>
<td>Window (15 var.)</td>
</tr>
<tr>
<td>ORC*</td>
<td>Join/Shuffle (12 var.)</td>
</tr>
<tr>
<td>Kafka</td>
<td>SQL UDF (7 var.)</td>
</tr>
<tr>
<td>FIX*</td>
<td></td>
</tr>
<tr>
<td>PCRE*</td>
<td></td>
</tr>
</tbody>
</table>
Database Templates RoadMap

Scan Engines

<table>
<thead>
<tr>
<th>Compression (5 var.)</th>
<th>JSON</th>
<th>CSV</th>
<th>AVRO*</th>
<th>Parquet</th>
<th>ORC*</th>
<th>Kafka</th>
<th>FIX*</th>
<th>PCRE*</th>
</tr>
</thead>
</table>

SQL PushDown Engines

- Project/Filter (40 var.)
- Sort
- Hashaggregate/
- Sortaggregate (16 var.)
- Window (15 var.)
- Join/Shuffle (12 var.)
- SQL UDF (7 var.)
Database Templates RoadMap

Scan Engines

- Compression (5 var.)
  - JSON
  - CSV
  - AVRO*
  - Parquet
  - ORC*
  - Kafka
  - FIX*
  - PCRE*

SQL PushDown Engines

- Project/Filter (40 var.)
- Sort
- Hashaggregate/
- Sortaggregate (16 var.)
- Window (15 var.)
- Join/Shuffle (12 var.)
- SQL UDF (7 var.)
# Database Templates RoadMap

## Bigstream SW Stack

### Scan Engines

- Compression (5 var.)
  - JSON
  - CSV
  - AVRO*
  - Parquet
  - ORC*
  - Kafka
  - FIX*
  - PCRE*

### SQL PushDown Engines

- Project/Filter (40 var.)
- Sort
- Hashaggregate/
- Sortaggregate (16 var.)
- Window (15 var.)
- Join/Shuffle (12 var.)
- SQL UDF (7 var.)
Conclusion

Bigstream enables SmartSSD
• Acceleration with zero code change
• Support for multiple big data platforms
• Seamless integration

SmartSSD enables
• Higher performance
• Greater scalability
• TCO savings
Thank You
Spark.sql("SELECT name FROM People WHERE age BETWEEN 25 AND 40")

Bigstream Hyper-acceleration with Multi-core CPUs and FPGAs

Acceleration occurs without code changes

Step 1
SQL Converted to Dataflow

Step 2
Dataflow Slicing

Step 3
Mapping of Execution
Transparent Hyper-acceleration at Runtime

Bigstream Dataflow Compiler

Bigstream Dataflow Runtime

SQL Query

Stage Acceleration Analyzer

Execute Stage Task

Dataflow μ-arch exists?

No

Run Stage on Native SW

Yes

Partial FPGA Reconfiguration

Dataflow μ-arch Parameter Configuration

Run Stage on FPGA

Last Stage?

No

Reuse Dataflow μ-arch?

No

Query Output

Hyper-acceleration Dataflow Detail
TPC-DS Ingest Heavy Single Node Results – Row Based
Example Walkthrough

val df = sparkSession.read.schema(schema).json(jsonRDD)

df.filter("country is not null and price is not null")
.filter(df("country").equalTo("USA"))
.groupBy("action_id").agg(count("price"), min("price"), max("price")).show()