SPADE: The System S Declarative Stream Processing Engine

B. Gedik, H. Andrade, K. Wu, P. Yu, and M. Doo
(SIGMOD. 2008)

Presented by Kenneth Lui (wckl2)
10th Nov 2015
Outline

● Background - Stream Processing Engine, System S
● Motivation
● System Design & Contribution - Programming Model, Optimization
● Example & Experiment Result
● Future Work
● Summary & Critical Analysis
Background
Stream Processing Engine

● “On-the-fly” processing of time ordered series of events or values
  ○ Low-Latency is key

● Data enter the system as “input stream”, get filtered, processed, aggregated etc. in the network of “computational elements” connected by streams

● Related Works
  ○ MillWheel (Google), Apache Storm (Twitter)
Stream Processing Use Cases

- Web log processing
- Sensor networks
- Real-time financial analysis
System S

- Large-scale, distributed data stream processing middleware and application development framework
- Applications organized as data-flow graphs
  - Sets of **Processing Elements (PEs)** connected by streams
  - PEs are distributed over the computing nodes
  - Each stream carries a series of **Stream Data Objects (SDOs)**
  - The PE ports and streams connecting them are **typed**
- Provide reliability, scheduling, placement optimization, security, fault tolerance etc.
Stream Processing Core (System S)

- Dataflow Graph Manager (DGM)
  - Define stream connections among PEs
- Data Fabric (DR)
  - Distributed data transport daemons
- Resource Manager (RM)
  - Makes global resource decisions for PEs and streams
- PE Execution Container (PEC)
  - Provide run-time context and security barrier

Figure 1: Key components of System S that provide services to run stream applications [18].
Motivation

Before SPADE, there were two ways of use System S...
Programming in PE API

- For experienced developer
- Write programs in **C++** or **Java** to interact directly with PEs
- Design configuration files to specify the topology of the data-flow graph (i.e. connect the PEs)
Working with Domain Specific Queries

- For less experienced developers
- Issue natural language-like domain-specific inquiries
- Inquiry Services (INQ) planner makes use of a repository of existing PEs to automatically create a data-flow graph
**SPADE - Declarative middle-ground**

- SPADE = Stream Processing Application Declarative Engine
- Declarative = Developers describe the problem rather than the steps to solve it
- Allow integration of User defined functions (UDFs) and Legacy Code
- Some manual tuning on deployment is possible
Figure 1: System S from an application developer’s perspective
System Design & Contribution
Code Generation Framework

- Compiler takes query specification written in SPADE’s intermediate language and produces these native parts in System S:
  - PE template
  - Node pools
  - PE topology
  - PE binaries
  - Job description (from System S Job Description Language Compiler)
Code Generation Framework

- SPADE compiler’s output is highly customized based on the system characteristics
  - Underlying network topology
  - Computer architecture
Figure 3: Spade’s code generation framework
Stream Processing Operators

- Functor
- Aggregate
- Join
- Sort
- Barrier - used as a synchronization point
- Puncto - generate punctuation for windowing
- Split
- Delay
Edge Adapters

● Source
  ○ Parsing
  ○ Tuple creation

● Sink
  ○ From streams to external data
  ○ E.g. file system, network
SPADE Programming Language

# %1 and %2 are the first and second parameters
#define NCNT min(%1,16) #* number of nodes to utilize *#
#define FCNT min(%2,30) #* number of days to analyze *#

[Application]
vwap # trace

[Typedefs]
typespace vwap

[Nodepools]
nodepool ComputingPool[16] := () # automatically allocated from available nodes

[Program]
/* Source data format:
 * 1 ticker:String, 8 volume:Float, 15 askprice:Float, 22 peratio:Float,
 * 2 ... */
for_begin @day 1 to FCNT # for each day

    stream TradeQuote@day(ticker:String, ttype:String, price:Float, volume:Float, askprice:Float, asksize:Float)
        := Source() ["file:///gpfs/ss/taq"+select(@day<10,"0@day","@day")+.csv", nodelays, csvformat] { 1, 5, 7-8, 15-16 }
            -> partition["mypartition_@day"], ComputingPool[mod(@day-1,NCNT)]

    stream TradeFilter@day(ticker: String, myvwap:Float, volume:Float)
        := Functor(TradeQuote@day) [ttype="Trade" & volume>0.0]
            { myvwap := price*volume }
            -> partitionFor(TradeQuote@day), ComputingPool[mod(@day-1,NCNT)]

    stream VWAPAggregator@day(ticker:String, svwap:Float, svolume:Float)
        := Aggregate(TradeFilter@day ) [ticker]
            { Any(ticker), Sum(myvwap), Sum(volume) }
            -> partitionFor(TradeQuote@day), ComputingPool[mod(@day-1,NCNT)]
stream `BargainIndex@day`(ticker:String, bargainindex:Float) := Join(VWAP@day ; QuoteFilter@day ) [[ticker]={ticker}, cvwap > askprice*100.0] { bargainindex := exp(cvwap-askprice*100.0)*asksize } -> partitionFor(TradeQuote@day), ComputingPool[mod(@day-1,NCNT)]

export stream `NonZeroBargainIndex@day`(schemaof(BargainIndex@day)) := Functor(BargainIndex@day) [bargainindex>0.0] {} -> partitionFor(TradeQuote@day), ComputingPool[mod(@day-1,NCNT)]

Null := Sink(NonZeroBargainIndex@day) ["file://Bargains@day.dat"]{} -> partitionOf(TradeQuote@day), ComputingPool[mod(@day-1,NCNT)]

for_end
User-Defined Operators

- Can make use of external libraries to implement domain-customized operations
- Allow converting legacy code to System S
- Support interfacing with external platforms
Advanced Features

- List Types and Vectorized Operations
- Flexible Windowing Schemes
  - Tumbling windows - fixed number of tuples
  - Sliding windows - expiration policy + trigger mechanism
  - Punctuation-based window boundaries
- Pergroup Aggregates and Joins
Compiler Optimizations

- Operator Grouping
- Execution Model
- Vectorized Processing
**Operator Grouping**

- Having multiple operators per PE is more efficient
- Reduce message transmission and queuing delays

*Figure 4: Example operator to PE mapping*
Execution Model

- To make use of multiple cores, SPADE creates multiple PE’s to be run on the same node.
- Multi-threading built-in operators were still under development.
Vectorized Processing

- Single-Instruction Multiple-Data (SIMD)
- E.g. Intel’s Streaming SIMD Extensions (SSE)
Operator Fusion

- Operators in the same PE are chained as depth-first function calls, without any queuing
- For thread-safe operators, SPADE supports multi-threading to cut short the main PE thread
  - May require locking
Two-phase learning-based Optimization

● First, compile the application in a special “Statistics Collection mode”
  ○ Application is run in this mode to collect metrics like CPU load and network traffic

● Then, compile the application for a second time
  ○ Optimizer uses statistics to guide operator grouping & fusion to come up with the PEs
Example & Experiment result
Bargain Index Computation

- Compute the bargain index (a scalar metric for stock trading analysis) for every stock symbol that appears in the source stream
- Source: Live stock data can be read directly from the IBM WebSphere Front Office (WFO)
- Sink: IBM DB2 Data Stream Edition – an extension of DB2 designed for persisting high-rate data streams
Bargain Index Computation

Figure 5: Bargain Index computation for all stock symbols
Experiment

- Process 22 days’ worth of ticker data for ≈ 3000 stocks with a total of ≈ 250 million trade and quote transactions
- ≈ 20GBs of data, sharded per file per day on the disk on IBM’s General Parallel File System (GPFS)
- Parallelize the processing by running 22 instances (PEs), one for each trading day, over 16 nodes in our cluster
**Issues with this experiment**

- All operators within the same query are packed into a single PE (i.e. single PE per day)
- No inter-node communication or cooperation
- Some resources are idle after ~23:07
- Compare with native System S API implementation?

---

**Figure 6:** Tuple ingestion rate for the parallel and distributed bargain index computation application, using 22 parallel queries distributed over 16 nodes.
Future Work
Future Work

- Visual development environment
- Domain-specific operator
  - (e.g. signal processing, stream data mining)
- Higher-level languages (Stream SQL, semantic composition framework)
  - A 2013 paper about “IBM Streams Processing Language (SPL)”
- Interoperability
  - Data ingestion and externalization with other platforms
Summary & Critical Analysis
Summary

● A declarative language which balances flexibility and barrier of entry
● Toolkit (compiler, stream operators)
● Bring stream processing to System S
Critical Analysis - System

- Partition and optimization happen at compile-time
- Does not adopt to capacity change (+/- nodes)
- No priority concept for the tuples
Critical Analysis - Paper

● Two-phase learning-based optimization is not discussed in depth
  ○ I am very curious about the development/deployment workflow here
  ○ It should compare the performance with/without this optimization

● No fault tolerance analysis
● Example & Evaluation not representative
Thank you!

Any questions?