What is driving the need for Stream Computing?

- **Growth in Data Volume**: Every day, 15 petabytes of new information are being generated. Data volume are expected to double every 2 years.
- **Growth in Variety**: 80% of new data growth is from non-relational and non-traditional data types like emails, documents, RFID feeds, multimedia etc.
- **Decision Quality: Bar has been raised**: CEOs are emphasizing the need to make faster and smarter decisions to reducing risk and enhance competitive advantages.
Stream Computing:
A new paradigm for ultra low latency and high throughput in-motion analytics

Continuous Ingestion  Continuous Queries /Analytics on data in motion
Evolution of Stream Computing

Reporting and human analysis on historical data

Analysis of historic data to improve business transactions

Real Time Analytic Processing (RTAP) to improve business response

Data at rest

Operational Databases

1968
Hierarchical database

1970
Relational database

1983
DB2 v1

Data Warehousing

2003

Stream Computing

OLTP

OLAP

RTAP
Traditional Analytics (OLAP/OLTP)
Traditional Analytics (OLAP/OLTP) Vs Stream Analytics (RTAP)

RTAP

Non-Traditional / Non-Relational Data Sources

In-Motion Analytics

Ultra Low Latency Results

Traditional / Relational Data Sources

Database

At-Rest Data Analytics

Results
Efficient Integration with Stored Data

- Integration enables enrichment and persistence
- Support for disk based databases and in-memory databases like solidDB
Stream computing for business advantage

- **Streaming analytic applications**
  - Fusing multiple input streams
  - Continuous queries
  - “Stateless” message processing
- **Simplified development**
- **Blazing speed and scalability**
  - Extreme data rates
  - Hundreds of data sources
- **Applications across industries**
  - Law enforcement, finance, space science, healthcare, telecommunications, manufacturing, environmental
Scalable stream processing: Why

• **Streams supports applications that require on-the-fly processing and analysis of streaming data**

• **Criteria: two or more of the following**
  
  • Messages are processed on their own (“stateless”) or in the context of limited data windows
  
  • sources deliver semi-structured or unstructured data, not easily processed using “Complex Event Processing” systems
  
  • sources vary in connection methods, data rates, and processing requirements, presenting integration challenges
  
  • processing needs exceed the resources of a single processing node
  
  • analysis and response are needed with sub-millisecond latency
  
  • data rates and data volumes are too great for store-and-mine approaches
IBM InfoSphere Streams v1.2 (since Feb 23, 2010)

Development Environment

- Streams Processing Language
- Eclipse IDE

Runtime Environment

- RHEL v5.3 or v5.4
- x86, 32- or 64-bit
- Up to 125 servers
- LLM (Infiniband) supported

Toolkits & Adapters

- Connectors to data sources
- Math & text functions
- Operator Library
- Mining, financial services toolkits

Streams Live Graph
Scalable stream processing:

**What**

- **Streams provides**
  - a programming model for defining data flow graphs consisting of *data sources* (inputs), *operators*, and *sinks* (outputs)
Scalable stream processing: What

• **Streams provides**
  
  • a programming model for defining data flow graphs consisting of *data sources* (inputs), *operators*, and *sinks* (outputs)
  
  • controls for fusing operators into processing elements (*PEs*)
Scalable stream processing: What

- **Streams provides**
  - a programming model for defining data flow graphs consisting of *data sources* (inputs), *operators*, and *sinks* (outputs)
  - controls for fusing operators into processing elements (*PEs*)
  - infrastructure to support the composition of scalable *stream processing applications* from these components
Scalable stream processing: What

- **Streams provides**
  - a programming model for defining data flow graphs consisting of **data sources** (inputs), **operators**, and **sinks** (outputs)
  - controls for fusing operators into processing elements (**PEs**)
  - infrastructure to support the composition of scalable **stream processing applications** from these components
  - deployment and operation of these applications across distributed **x86 processing nodes**, when scaled-up processing is required
Streams application physical view

- A collection of interconnected processing elements
- Each processing element, or PE, is a collection of one or more fused operators
Streams application logical view

- A directed data flow graph
  - Nodes: operators
  - Edges: streams
Streams Studio Integrated Development Environment
Streams Studio is highly extensible and visual

Highly Visual and Intuitive

• **SPADESight**
  - Tool to view the analytics flow graph

• **StreamSight**
  - Tool to view runtime statistics and data flow
  - Integrated with distributed debugger

Highly Flexible and Extensible

• **Wide range of data adapters**
  - Intuitive framework to develop additional adapters

• **Assortment of powerful stream operators, text and math functions**
  - Additional custom operators can be written in SPADE or C++/Java

Adapters

Files  DB2  z/OS  solidDB  WebSphere  Front Office 3.0  TCP Sockets
Monitor applications, track and debug data flow
Streams Processing Language: built for streaming data

Streams Processing Language is a stream-centric and operator-based language for declaring stream processing applications

- Type-generic declarative data stream processing operators
- User-defined operators to wrap legacy code
- Procedural constructs to control the composition and coalescing level of operators

Declarative at the operator level
Procedural at the composition level
Streams Processing Language – Highlights

• A toolkit of *Stream Processing Operators*
  • All stream relational operators and more…
  • Operate on data *tuples*

• Flexible *Edge Adapters*
  • To read/write from/to external data sources

• Support for *User-Defined Operators* (UDOPs)

• Support for *User Defined Built-in Operators* (UBOPs)

• Procedural constructs
  • To create distributed and parallel flow graphs

• *Vectorized* operations
  • Supports lists and matrices of primitive types

• Knobs to influence placement, partitioning, execution model, etc.
Built-in operators (BIOPs)

- **The Aggregate operator** is used for grouping and summarization of incoming tuples.
- **The Split operator** is used for splitting a stream into multiple output streams.
- **The Join operator** is used for correlating two streams.
- **The Punctuator operator** is for inserting punctuation marks in streams.
- **The Aggregate operator** is used for grouping and summarization of incoming tuples.
- **The Sort operator** is used for imposing an order on incoming tuples in a stream.
- **The Function operator** is used for performing tuple-level manipulations.
- **The Delay operator** is used to “artificially” slowdown a stream.
- **The Barrier operator** is used as a synchronization point.
Streams Operators

- **Examples of Operators**
  - **Functor**: Add attributes, remove attributes, filter tuples, map output attributes to a function of input attributes
  - **Aggregate**: Window-based aggregates, with group by
  - **Punctor**: Punctuation mark to delimit windows
  - **Join**: Window-based binary stream join
  - **Sort**: Window-based approximate sorting
  - **Barrier**: Synchronize multiple streams

- **Examples of Adapters**
  - **File**: Read/write from/to files
  - **TCP**: Read/write from/to TCP sockets
  - **UDP**: Read/write from/to UDP sockets
  - formats: (binary, csv) / (compressed, uncompressed)
  - **WFO**: Read from WebSphere Front Office
  - **ODBC**: Read/Write to Relational DBMS
  - **solidDBEnrich**: solidDB Accelerator API

- **User-Defined Operators**
  - Wrap external libraries
  - Implement non-relational operators
Extending the language

- **User-Defined Functions (UDFs)**
  - Use in operator expressions (e.g., Functor)

- **User-Defined Operators (UDOPs)**
  - Integrating legacy code
  - Using “proprietary” libraries
  - UDOPs are black boxes

- **User-defined Built-In Operators (UBOPs)**
  - The Streams programming model can be extended with additional toolkits
    - Data mining
    - Financial math
  - A template-based programming environment is provided
  - UBOPs are type-generic and transparent
Compiler Framework

- **Operator Fusion**
  - Fine-granularity operators
  - From small parts, make a big one that fits

- **Code generation**
  - Actual code must match the underlying runtime environment
    - Number of cores
    - Interconnect characteristics
    - Architecture-specific instructions

- **Compiler-based optimization**
  - Driven by automatic profiling
  - Driven by incremental learning of application characteristics
A simple example

[Application]
SourceSink trace

[Typedefs]
typespace sourcesink
typedef id_t Integer
typedef timestamp_t Long

[Program]
// virtual stream schema declaration
vstream Sensor (id : id_t, location : Double, light : Float, temperature : Float, timestamp : timestamp_t)

// a source stream is generated by a Source operator -- in this case tuples come from an input file
stream SenSource ( schemaFor(Sensor) )
 := Source( ) [ “file:///SenSource.dat” ] {}

// this intermediate stream is produced by an Aggregate operator, using the SenSource stream as input
stream SenAggregator ( schemaFor(Sensor) )
 := Aggregate( SenSource <count(100),count(1),perGroup> ) [ id . location ]
 { Any(id), Any(location), Max(light), Min(temperature), Avg(timestamp) }

// this intermediate stream is produced by a Functor operator
stream SenFunctor ( id: Integer, location: Double, message: String )
 := Functor( SenAggregator ) [ log(temperature,2.0)>6.0 ]
 { id, location, “Node “+toString(id)+ “ at location “+toString(location) }

// result management is done by a Sink operator -- in this case produced tuples are sent to a socket
Null := Sink( SenFunctor ) [ “udp://192.168.0.144:5500/” ] {}
Example: development workflow in Streams Studio

1. Create/Import Streams Project
2. Type application specification using Streams Studio editor
3. Build / launch as standalone executable
4. Run standalone application using stream debugger (SDB)
5. Debug UDOP code using source code debugger (GDB)
6. Rebuild / launch on Streams runtime

Streams runtime

(7) Use debug wrapper to run SDB in PE
InfoSphere Streams Runtime

Streams compiler

Job manager

Streams Data Fabric

Source

PE

Sink

Physical Network

x86 Node

Processing Element Container

Processing Element Container

Processing Element Container

Processing Element Container

Processing Element Container
General architectural features (1)

• **Runtime system logical view and state management**
  - Administration Console
    - Jobs, PEs, hosts, runtime services
  - Streams Live Graph
    - Data-flow graph monitoring
    - In Streams Studio

• **Authorization and Authentication**
  - User isolation
  - Inter-component mutual authentication
General architectural features (2)

- **Scheduling**
  - Initial placement decision for jobs
    - Based on system load
    - No relocation of PEs unless terminated
  - Semidynamic instance load balancing
    - If multiple transient jobs
- **System physical view state management**
  - Health and load of the hosts supporting the runtime environment
  - System-wide job and host control from single node
Fault-tolerance features (1)

• **Middleware fault-tolerance capabilities**
  • Cover all runtime management services
  • Common reliability infrastructure
    • Persists the internal state of a service to an IBM DB2 database
    • Fault tolerance at the inter-service protocol
  • Crashed services can be relocated and restarted
    • Manual (or scripted)
    • Minimal disturbance to the overall middleware
Fault-tolerance features (2)

- **Application-level fault-tolerance capabilities**
  - Processing elements can be moved and restarted in reaction to software and hardware failures
    - Automatic if failure is in PE itself
    - Manual intervention needed if host fails
  - Checkpoint capabilities available in the programming model

- **Fault tolerance does NOT mean data recovery**
  - Fundamental distinction with databases
Performance Results

• 100-node cluster
  • System test validated on 100 nodes

• Basic throughput (thousands of 256-byte tuples/second – ktps)
  • Streams 1.0 32-bit: 422 ktps
  • Streams 1.0 64-bit: 418 ktps

• 7-day-long run
  • CPU utilization over 90% on all nodes

• Sustained data rate of 1 million tuples / second – Mtps)
  • 64-bit, 6-node system can process 1.3 Mtps
  • Release 1.2, using Infiniband, can do this with 2-node system

• Latency well under 1 millisecond (1 ms = 1000 µs)
  • Streams 1.0 32-bit: 116 µs
  • Streams 1.0 64-bit: 107 µs
‘Smart’ applications are fast emerging

- Neonatal Care
- Trading Advantage
- Environment
- Law Enforcement
- Radio Astronomy
- Telecom
- Manufacturing
- Smart Traffic
- Fraud Prevention
Financial Services

• **Identify trading opportunities**
  - Based on weather-related and other external factors
  - Integration of many disparate data feeds

• **Market surveillance**
  - Analyze trading patterns for evidence of fraud
  - High volume, low latency

• **Equities trading**
  - Sample application from Financial Services Toolkit
  - Flexible deployment, application evolution
Web Zero platform

Capture weather sensor data, analyses hurricane predicted path

Estimate impact on portfolios

Recommendations Based on Hurricane Forecast

Compute portfolio market indicators (low latency)

Make recommendations and notify

Capture market data (high volume)

System S platform

DHTML Result

Real-time projections of hurricane path

Dynamically updated risk assessment for assets in projected path

Correlate combined risk and trade VWAP to determine buy/sell recommendations
Law Enforcement and Security – Federal Government

Streams of information including video surveillance, wire taps, communications, call records, etc.

Millions of messages per second with low density of critical data

Identify patterns and relationships among vast information sources

"The US Government has been working with IBM Research since 2003 on a radical new approach to data analysis that enables high speed, scalable and complex analytics of heterogeneous data streams in motion. The project has been so successful that US Government will deploy additional installations to enable other agencies to achieve greater success in various future projects" - US Government
Streams for Real-time Geomapping

Multiple GPS Data Sources
- 350-400K probe points per second per source
- Map probe point to nearest poly-line (Map)
- 200 million – 1 billion poly-lines
- 2 level grid decomposition based search

14 Blade servers
- 2X Dual-Core Xeon 5160
- 16 GB RAM
- 4 data prep, 10 mapping servers

Performance
- 941,000 probes/sec for 1 Billion poly-lines

Real-time location profile
Traffic Management for Sustainability and Efficiency

• Multimodal Data Streams
  • GPS
  • Cell-phones (location tracking)
  • Public Transport (bus, docking)
  • Pollution measurements
  • Weather Conditions (including road conditions)
  • Optical traffic flow detectors
  • Travel time data based on plate recognition
  • Induction loop detector data
  • Accidents in network as they are recorded
  • Road closures (road work, etc)
  • Still pictures from road cameras

Only 4 x86 Blade servers to process 250,000 GPS probes per second, maps of 630,000 line segments

• Real Time Traffic Monitoring & Information
• (Multimodal) Travel Planner
Predictive Analytics using InfoSphere Streams in a neonatal ICU helps detect life-threatening conditions up to 24 hrs earlier

• **Real Time analytics and correlations on physiological data streams**
  • Blood pressure, Temperature, EKG, Blood oxygen saturation etc.,

• **Early detection of the onset of potentially life threatening conditions**
  • Up to 24 hours earlier than current medical practices
  • Early intervention leads to lower patient morbidity and better long term outcomes

• **Technology also enables physicians to verify new clinical hypotheses**
Next Steps:
ibm.com/software/data/infosphere(streams
ibm.com/developerworks/wikis/display(streams/Home
ibm.com/developerworks/wikis/display(streams/Demos