Language & System Support for Efficient State Sharing in Distributed Stream Processing Systems

IBM T. J. Watson Research Center
Outline

- Motivation
- Design considerations
- Detailed design
- Implementation & evaluation
- Summary
What is the need for state sharing in stream processing systems?

- **Control Variables**
  - In a long running System S application, a user may want to modify the behavior of some operators at runtime
  
  - Examples: *filtering threshold, routing behavior, lookup tables etc.*
What is the need for state sharing in stream processing systems?

- A shared runtime repository of interesting events
  - Operators collaborate to detect and follow-up on interesting events observed by the application
  - Examples: *intrusion detection*
Motivation

Why not use System S to propagate updates? – Control spaghetti
Efficient state sharing in stream processing systems - Why is it hard?

- **Ease-of-use & Flexibility**
  - Many System S users are domain experts and/or analysts with sufficient but not a deep understanding of issues related to distributed shared state.
  - System S is used for a range of applications (e.g. healthcare, telecommunications, finance, etc.) that have very different expectations from shared state implementation.

- **Scalability, High-Performance & Fault-Tolerance**
  - The state sharing mechanism should be such that it limits the impact on the scalability and performance of the System S application. Also, the exposure of the user to issues like fault-tolerance of the shared state should be minimized.

- **Relaxed Consistency Guarantees**
  - Given the fact that many System S applications do not require atomic consistency for access to the shared state, the state sharing mechanism should be able to exploit the relaxed consistency requirements for enhanced scalability and/or performance.
Outline

- Motivation
- Design considerations
- Detailed design
- Implementation & evaluation
- Summary
Ease-of-use & Flexibility

SPADE language constructs

What were we thinking?

Provide flexibility to users while maintaining the ease of use
Ease-of-use & Flexibility: SPADE language constructs

\[ \text{sharedVarDef} ::= \text{sharedVarModifier}^* \text{ type ID ( = expr )? sharedVarConfigs} \]
\[ \text{sharedVarModifier} ::= \text{public} | \text{static} | \text{mutable} \]
\[ \text{sharedVarConfigs} ::= \text{;} | \text{ { ‘config’ configuration+ ‘} } \]

- **public** – may be used from anywhere in the system
- **static** – all instance of the operator defining the shared variable will share the same copy
- **mutable** – can be modified
- **configuration** – name-value pair
Example usage

composite CompositeWithSharedVariables(Output out; Input in) {
  var int32 s_thresh = 10;

  public static mutable map<string8, int32> s_map {
    config lifetime : eternal;
    consistency : causal;
    sizeHint : 1024 * 128 * 128;
  }

  graph stream<In> X = ClassifierX(In) { param cMapX : s_map; }
  stream<In> Y = ClassifierY(In) { param cMapY : s_map; }
  stream<In> Out = Functor(X, Y) { param filter : x > s_thresh; }
}
Once the shared variables are defined in a SPADE program…

Compile Time

Runtime
Compile Time

Shared variable data types

What were we thinking?

State sharing should be transparent

Shared Variable data types should be oblivious of the transport and/or protocol
A view of Shared Variable from 30,000 feet

Detailed design

Shared Variable Client

SPADE Operator

Transport

Shared Variable Server

Shared Variable
A view of Shared Variable from 20,000 feet

- **Stub Side Data Type**
- **Stub Side Transport**
- **Server Side Transport**
- **Server Side Data Type**

**Shared Variable Client**

**SPADE Operator**

**Shared Variable Server**

**Shared Variable**
Client / Server side data types and `invoke` interface

SVInteger

void invoke(int methodIndex, Buffer inParams, Buffer outParams);

Stub Side Data Type

Server Side Data Type

int32 get();
void set(int32);
int32 add(int32);
int32 subtract(int32);
int32 multiply(int32);
int32 divide(int32);
int32 modulo(int32);

+ - * /
% += -=
*=/ %=
++ --
& | ^ ~
Example Shared Variable data type

- Server side data type
  \[
  \text{SVIntegerServer}\langle T \rangle \\
  [T = \text{int8, int16, int32, int64}]
  \]

- Stub side data type
  \[
  \text{SVIntegerClient}\langle T, I \rangle \\
  [T = \text{int8, int16, int32, int64}] \\
  [I = \text{SVBasicInterfaceCorbaClientImpl}, \ldots]
  \]
Compile time
Shared Variable transport and protocol

What were we thinking?
Should be usable with any compatible data client / server
A view of Shared Variable from 20,000 feet
Shared Variable transport

Stub Side Transport

Server Side Transport

Transport

Implements invoke Interface

Calls invoke Interface
Shared Variable protocol implementation
Example Shared Variable transport and protocol

- **Server side transport and protocol type**
  
  `SVBasicInterfaceCorbaServerImpl<T>`
  
  \[ T = SVInteger, SVFloat, \ldots \]

- **Client side transport and protocol type**
  
  `SVBasicInterfaceCorbaClientImpl`
Compile time

Shared variable servers, clients and the SVDL – putting it together
Shared Variable stub

1. Operator utilizes the stub side data type assuming that it is a regular data type.
e.g.
++i

2. The stub side data type serializes the parameters and translates the operation to an invoke call on the stub side transport
e.g.
Buffer inParams, outParams;
params << 1;
stub->invoke(ADD, inParams, outParams);

3. The stub side transport, if needed, marshals the data to transport specific format makes a remote call to the transport server at the other end.
e.g.
remote->invoke(ADD, inParams, outParams, exception);
Shared Variable server

1. Server side transport receives a call from the client side transport.

2. The server side transport, if needed, unmarshals the data to Buffer and calls invoke on the data server.
   
   e.g.
   
   `server->invoke(ADD,inParams,outParams);`

3. The server side data type deserializes the parameters and performs the appropriate operation on the contained data element

   e.g.
   
   `inParams >> temp;`
   `outParams << this->add(temp);`
Shared Variable stub and server example

SVIntegerClient<int32, SVBasicInterfaceCorbaClientImpl>
SVBasicInterfaceCorbaServerImpl<SVIntegerServer<int32>>

Shared Variable Client

Transport

Shared Variable Server
Shared variable description language (SVDL)

- Describes the composition of a shared variable

- Various constructs
  - Base variable
    - refers to a shared variable server, needs dll and location
  - Variable group
    - a protocol governed group of base variable, shared variable and / or variable group
  - Shared variable
    - contains a base variable or a variable group and has a name

- Is part of the Application Definition Language and is loaded by the Shared variable daemon at deployment time
SVDL example

```
<sharedVariable>
  <name>A</name>
  <variableGroup>
    <protocol>Atomic</protocol>
    <baseVariable>
      <dll>/users/omega/abc.so</dll>
      <location>192.168.2.101</location>
    </baseVariable>
    <baseVariable>
      <dll>/users/omega/abc.so</dll>
      <location>192.168.2.102</location>
    </baseVariable>
  </variableGroup>
</sharedVariable>
```
Implementation & Evaluation

- Besides the implementation of data types, we have a transport implementation based on CORBA.

- We have implemented 4 protocols – Atomic Master-Slave, Atomic Master-Slave with Buffer, Causal and Partitioned protocol. Other implementations will follow.

- The reported experiments were conducted on a 2 x dual core machines @ 3.0 GHz with 8 GB RAM
Comparison between performance of AMS and AMSB
Comparison between performance of AMSB and Causal protocol
Summary & Future Work

- Shared variables in System S attempt to exploit configuration parameters to code generate a customized implementation for higher performance.

- Maintaining conformity to SPADE’s native data types makes it simple to program using Shared Variables.

- Initial scalability and performance results seem to be very promising.

- Work is ongoing to determine the best heuristic that translates configuration parameters (e.g. readsPerSecond, writesPerSecond, consistency, etc.) to the most appropriate generated code for shared variables.

- Work is ongoing to incorporate the dependency between various clients (operators) into the Shared Variable consistency model.
Thank You!