Taverna, reloaded

Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Alex Nenadic,
Ian Dunlop, Alan Williams, Carole Goble
School of Computer Science
University of Manchester, UK

Wei Tan
Argonne National Laboratory
Argonne, IL, USA

SSDBM
Heidelberg, Germany
June 30 - July 2, 2010

Tom Oinn
EBI, UK
Goal: perform cancer diagnosis using microarray analysis
- learn a model for lymphoma type prediction based on samples from different lymphoma types

source: caGrid

http://www/myexperiment.org/workflows/746
Goal: perform cancer diagnosis using microarray analysis - learn a model for lymphoma type prediction based on samples from different lymphoma types

source: caGrid
Goal: perform cancer diagnosis using microarray analysis
- learn a model for lymphoma type prediction based on samples from different lymphoma types

source: caGrid

http://www/myexperiment.org/workflows/746
Goal: perform cancer diagnosis using microarray analysis
- learn a model for lymphoma type prediction based on samples from different lymphoma types

source: caGrid

http://www/myexperiment.org/workflows/746
Goal: perform cancer diagnosis using microarray analysis
- learn a model for lymphoma type prediction based on
samples from different lymphoma types

source: caGrid

http://www/myexperiment.org/workflows/746
• Taverna for data-intensive science
  – architectural goals... and how they are being achieved

• Performance figures on benchmark workflows
• **Scalability**
  – size of input data / size of input collections

• **Configurability**
  – each processor in the workflow individually tunable to adjust to the underlying platform

• **Extensibility**
  – plugin architecture to accommodate specific service groups
    • caGrid, BioMoby
  – extensible set of operators that define the semantics of the model
Opportunities for parallel processing

- intra-processor: implicit iteration over list data
- inter-processor: pipelining

\[ [id1, id2, id3, \ldots] \]
Opportunities for parallel processing

- intra-processor: implicit iteration over list data
- inter-processor: pipelining
Opportunities for parallel processing

- intra-processor: implicit iteration over list data
- inter-processor: pipelining
Configurable processor behaviour

- allocates concurrent threads to list elements
- captures service invocation events
- persistent server error: tries each available service implementation in turn
- transient errors: retries one service invocation

Interacts with underlying activity
- service operation
- shell script execution

Request

Dispatch stack

Default dispatch stack
- Parallelise
- Provenance
- Error bounce
- Failover
- Retry
- Stop/pause
- Invoke

Response

Workflow Inputs
- EXP_ID
- BeanShell
- searchForHybridizations
- BeanShell2
- getDataSet_input
- getDataSet

Workflow Outputs
- DataSet
Extensible processor semantics

Default dispatch stack
- Parallelise
- Provenance
- Error bounce
- Failover
- Retry
- Stop/pause
- Invoke

Dispatch stack

Request → Response
Extensible processor semantics

Enhancing dataflow model with limited form of loops to model busy waiting (asynchronous service interaction)
Extensible processor semantics

Enhancing dataflow model with limited form of loops to model busy waiting (asynchronous service interaction)

Busy waiting using an explicit workflow pattern
Extensible processor semantics

Enhancing dataflow model with limited form of loops to model busy waiting (asynchronous service interaction)

Busy waiting using an explicit workflow pattern

Busy waiting using an loop processor
Performance experimental setup

- previous version of Taverna engine used as baseline
- objective: to measure incremental improvement

Parameters:
- byte size of list elements (strings)
- size of input list
- length of linear chain

main insight: when the workflow is designed for pipelining, parallelism is exploited effectively
list size: 1,000 strings of 10K chars each
no intra-processor parallelism (1 thread/processor)
list size: 1,000 strings of 10K chars each
no intra-processor parallelism (1 thread/processor)
Available processors pool

pipelining in T2 makes up for smaller pools of threads/processor
Separation of data and process spaces ensures scalable data management

varying data element size: 10K, 25K, 100K chars

**Taverna 2.1 - different iteration item sizes.**
Summary

- Taverna 2 re-engineered for scalability on data-intensive pipelined dataflows
- separation of data and process spaces
- generic stack pattern for principled extensions
  - limited while loop, if-then-else
  - provenance capture
  - ... open plugin architecture accommodates further extensions

For further details and a nice chat: please visit our poster!!