Synergy: Quality of Service Support for Distributed Stream Processing Systems

Thomas Repantis
Department of Computer Science & Engineering
University of California, Riverside

trep@cs.ucr.edu
http://www.cs.ucr.edu/~trep/
Research Contributions

- Distributed Stream Processing Systems
  - Sharing-Aware Component Composition [Middleware’06, TPDS'08 (rev.)]
  - Load Prediction and Hot-Spot Alleviation [DSN’08, DBISP2P’07]
  - Replica Placement for High Availability [DEBS'08]

- Management of Large-Scale, Distributed, Real-Time Applications
  - Adaptation to Resource Availability [IPDPS’05]
  - Fair Resource Allocation [ISORC’06, WPDRTS’05]

- Peer-to-Peer Systems
  - Adaptive Data Dissemination and Routing [MDM’05]
  - Decentralized Trust Management [MPAC’06]

- Software Distributed Shared Memory Systems
  - Data Migration [Cluster’05, Cluster’04]
  - Replication in Distributed Multi-Tier Architectures [IBM’07]
  - Collaborative Spam Filtering [Intel’06]
  - Distributed Logging for Asynchronous Replication [HP’05]
On-Line Data Stream Processing

Network traffic monitoring for intrusion detection

Analysis of readings coming from sensors or mobile robots

Click stream analysis for purchase recommendations or advertisements

Customization of multimedia or news feeds

Thomas Repantis
Distributed Stream Processing System

High volume data streams (sensor data, financial data, media data)

Real-time online processing functions/Continuous query operators

Filter  Correlation  Clustering  Aggregation

Extracted result streams
Streams are processed online by components distributed across hosts.
Data arrive in large volumes and high rates, while workload spikes are not known in advance.
Stream processing applications have QoS requirements, e.g., e2e delay.

Thomas Repantis
QoS for Distributed Stream Processing Applications

Our goal: How to run stream processing applications with QoS requirements, while efficiently managing system resources

- Share existing result streams
- Share existing stream processing components
- Predict QoS violations
- Alleviate hot-spots
- Maximize availability

Benefits

- Enhanced QoS provision
- Reduced resource load

Challenges

- Concurrent component sharing
- Highly dynamic environment
- On-demand stream application requests
- Scale that dictates decentralization

Thomas Repantis
Roadmap

- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Component Composition
  - Composition Protocol
  - Component and Stream Sharing
- Load Balancing
  - Hot-Spot Prediction
  - Hot-Spot Alleviation
- High Availability
  - Replica Placement
- Conclusion
- Demo
Synergy Middleware

- A middleware managing the mappings:
  - From application layer to stream processing overlay layer
  - From stream processing overlay layer to physical resource layer
Metadata Layer Over a DHT

- Decouples stream and component placement from their discovery.
- Stream and component names are hashed in a DHT.
- DHT maps the hashed names to nodes currently offering the specified stream or component.

Thomas Repantis
Synergy Node Architecture

- **Application Composition and QoS Projection** instantiate applications
- **Replica Placement** places components
- **Load Balancing and Load Prediction** detect hot-spots
- **Migration Engine** alleviates hot-spots
- **Monitor** measures processor and bandwidth
- **Discovery** locates streams and components
- **Routing** transfers streaming data
Component Composition

Query Plan + QoS Requirements

Synergy Middleware

Application Component Graph

Thomas Repantis
Composition Probes

- Carry query plan, resource, and QoS requirements
- Collect information about:
  - Resource availability
  - End-to-end QoS
  - QoS impact on existing applications

Thomas Repantis
Composition Protocol

**Input**
- Query Plan
  - Stream application template
  - QoS requirements
  - Resource requirements

**Output**
- Application Component Graph
  - Satisfy QoS and resource requirements
  - Reuse streams and components without QoS violations
  - Achieve load balancing
Composition Selection

- All successful probes returning to source have been checked against constraints on:
  - Operator functions
  - Processing capacity
  - Bandwidth
  - QoS

- The most load balanced one is selected among all qualified compositions by minimizing:

\[
\phi(\lambda) = \sum_{v_i \in V_{\lambda}, o_i \in \xi} \frac{p_{o_i}}{r p_{v_i} + p_{o_i}} + \sum_{l_j \in \lambda, s_j \in \xi} \frac{b_{s_j}}{r b_{l_j} + b_{s_j}}
\]
Component Sharing

- QoS Impact Projection Algorithm
  - All existing and the new application should not exceed requested execution time:

\[ \delta + \hat{t} \leq q_t \]

- Impact estimated using a queueing model for the execution time:

\[
\delta = \hat{t}' - \hat{t} = \frac{\tau_{c_i}}{1 - (p_{v_i} + p_{c_i})} - \frac{\tau_{c_i}}{1 - p_{v_i}}
\]
Stream Sharing

Maximum Sharing Discovery Algorithm

- Breadth first search on query plan to identify latest possible existing output streams
- Backtracking hop-by-hop, querying the metadata layer

Thomas Repantis
Experimental Setup

- PlanetLab multi-threaded prototype of about 35000 lines of Java running on 88 PlanetLab nodes
- Simulator of about 8500 lines of C++ for 500 random nodes of a GT-ITM topology of 1500 routers
- 5 replicas of each component
- Synergy vs Random, Greedy, and Composition

Thomas Repantis
Composition Performance

Stream reuse improves end-to-end delay by saving processing time and increases system capacity

Thomas Repantis
Composition Overhead

Stream reuse decreases probing overhead and setup time
Performance on Simulator

End-to-end delay scales due to stream reuse and QoS impact projection

Thomas Repantis
Sensitivity on Simulator

Synergy performs consistently better, regardless of QoS strictness or query popularity

*Thomas Repantis*
Projection Accuracy

Pessimistic projections for low rate segments may cause conservative compositions but no QoS violations

Thomas Repantis
Roadmap

- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Component Composition
  - Composition Protocol
  - Component and Stream Sharing
- Load Balancing
  - Hot-Spot Prediction
  - Hot-Spot Alleviation
- High Availability
  - Replica Placement

- Conclusion
- Demo
Application-Oriented Load Management

- System hot-spots: Overloaded nodes
- Application hot-spots: QoS violations
  - Sensitive hot-spot detection
    - Triggered even when underloaded, if stringent QoS
  - Fine-grained hot-spot alleviation
    - Only suffering applications migrate
- Proactively prevent QoS degradation
Predicting QoS Violations

Calculate slack time \( t_s \) on every component based on execution time \( t_e \) and communication time \( t_c \)

\[
 t_s(i) = qt - \left( \sum_{j \in 1 \ldots i-1} t_c(j \rightarrow j+1) + \sum_{j \in 1 \ldots i-1} t_e(j) + \sum_{i \ldots v-1} t_c(j \rightarrow j+1) + \sum_{j \in i \ldots v} \hat{t}_e(j) \right) > 0
\]
Execution Time Prediction

- Linear regression to bind execution time $t_e$ and total rate $r_t$

$$
\hat{t}_e = a + b \cdot \hat{r}_t
$$

$$
a = \bar{t}_e - b \cdot \bar{r}_t
$$

$$
b = \frac{\sum_{j=1\ldots k} (r_{t(j)} - \bar{r}_t) \cdot (t_{e(j)} - \bar{t}_e)}{\sum_{j=1\ldots k} (r_{t(j)} - \bar{r}_t)^2}
$$
Rate Prediction

▶ Auto-correlation

\[
\hat{r}_k = \frac{\arg_m \max R(k)}{\arg_m \max R(k-1)} \cdot r_{k-1} = \frac{r_k(m)}{r_{k-1}(m)} \cdot r_{k-1}
\]

▶ Cross-correlation (Pearson Product Moment)

\[
R_i = \frac{\sum_{j=1 \ldots (k-1)} (r_{j(i)} - \bar{r})(r_j - \bar{r})}{\sqrt{\sum_{j=1 \ldots (k-1)} (r_{j(i)} - \bar{r})^2 \sum_{j=1 \ldots (k-1)} (r_j - \bar{r})^2}}
\]

Thomas Repantis
Decentralized Load Monitoring

- DHT maps component names to the loads of peers hosting them
- Peers detect overloads and imbalances between all hosts of a component

- Load updates pushed when intervals change
- Overlapping intervals absorb frequent changes

Thomas Repantis
Alleviating Hot-Spots via Migration

Filter from B to A
B->A: Migration Request
A: QoS Projection
A->B: Migration Reply
B->E: Migration Update Request
B->F: Migration Update Request
E->B: Migration Update Reply
F->B: Migration Update Reply

Thomas Repantis
Hot-Spot Prediction and Alleviation

Average prediction error 3.7016%
Average prediction overhead 0.5984ms

Thomas Repantis
Hot-Spot Prediction and Alleviation

Average one migration every three applications
Average migration time 1144ms

Thomas Repantis
QoS Improvement

As load increases the benefits of hot-spot elimination become evident

*Thomas Repantis*
Roadmap

- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Component Composition
  - Composition Protocol
  - Component and Stream Sharing
  - Load Balancing
    - Hot-Spot Prediction
    - Hot-Spot Alleviation
  - High Availability
    - Replica Placement
- Conclusion
- Demo
Component Replication

Source: s1
- c11
  - s2
  - s3
  - s4

Destination: s6
- c41
  - s4
  - s5

s2+s3
- c31
  - c21
  - c42
  - c32

Thomas Repantis
Component Replica Placement

- Maximize availability of composite applications
  - Optimal: Place complete graph on each node
- Respect node resource availability
  - Processing capacity
  - Network bandwidth
- Maximize application performance
  - Inter-operator communication cost (between primaries)
  - Intra-operator communication cost (between primaries and backups)
Placement for High Availability

Availability decreases with larger graphs and increases with higher concentration

Thomas Repantis
Distributed Placement Protocol

Closest used candidates

Thomas Repantis
Replica Placement

Increase availability and performance
5539ms to gather latencies for 30 nodes

Thomas Repantis
Related Work

- System S: IBM stream processing middleware
- SBON, SAND, IFLOW: Component placement
- Borealis, Flux, PeerCQ: Load balancing
- Borealis, TelegraphCQ: Load shedding
- Borealis, Flux: Fault tolerance
- SpiderNet, sFlow: Component composition
Conclusion

- Synergy: QoS-Enabled Distributed Stream Processing System
  - Component Composition
    - Fully distributed composition protocol
    - Reuse existing streams and components
  - Load Balancing
    - Predict QoS violations
    - Alleviate hot-spots using migration
  - High Availability
    - Place component replicas
- Future work
  - Efficient and consistent replication
  - Adaptive topology management
  - Secure composite applications
Demo

- Monitor source-destination pairs in top 5% of total traffic over last 20 minutes [Stream Query Repository]

- TCP traffic trace, LBL, 2 hours, 1.8 million packets [Internet Traffic Archive]

<table>
<thead>
<tr>
<th>timestamp</th>
<th>sourceIP</th>
<th>destinationIP</th>
<th>sourcePort</th>
<th>destinationPort</th>
<th>size</th>
</tr>
</thead>
</table>
GUI Settings

The application to run: netmon

Number of services in an application: 3

Maximum requested rate from the source to the destination of the application in kbps: 50

Number of data units to send before we exit: 10000000

The size of the payload of every data unit: 250
GUI Application
GUI Execution

![Image of GUI Execution window]

<table>
<thead>
<tr>
<th>Application List</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.348691 44 29 1032</td>
</tr>
<tr>
<td>2.945475 54 55 1136</td>
</tr>
<tr>
<td>3.313864 4 23 1066</td>
</tr>
<tr>
<td>5.234655 4 23 1818</td>
</tr>
<tr>
<td>5.707645 40 41 2048</td>
</tr>
<tr>
<td>7.633051 40 41 1024</td>
</tr>
<tr>
<td>8.861617 40 41 2048</td>
</tr>
<tr>
<td>9.503625 40 41 4096</td>
</tr>
<tr>
<td>10.783522 29 58 5632</td>
</tr>
<tr>
<td>11.843829 4 23 1056</td>
</tr>
<tr>
<td>13.074514 4 23 1828</td>
</tr>
<tr>
<td>15.232693 4 23 1017</td>
</tr>
<tr>
<td>16.314291 4 53 2272</td>
</tr>
<tr>
<td>17.105083 44 29 1752</td>
</tr>
<tr>
<td>18.462050 4 52 2595</td>
</tr>
<tr>
<td>18.716051 29 30 2746</td>
</tr>
<tr>
<td>20.350927 4 23 1541</td>
</tr>
<tr>
<td>20.647518 4 52 3484</td>
</tr>
<tr>
<td>22.177906 4 23 1721</td>
</tr>
<tr>
<td>23.372508 29 30 1769</td>
</tr>
<tr>
<td>23.916157 98 99 13318</td>
</tr>
<tr>
<td>25.377907 98 99 109569</td>
</tr>
<tr>
<td>26.034145 98 99 68612</td>
</tr>
<tr>
<td>27.631830 4 53 3054</td>
</tr>
<tr>
<td>27.646943 40 41 4100</td>
</tr>
</tbody>
</table>

**Thomas Repantis**

44/45
Acknowledgements

- Prof. Vana Kalogeraki, UC Riverside
- Prof. Xiaohui Gu, NCSU (formerly IBM Research)
- Yannis Drougas, UC Riverside
- Bilson Campana, UC Riverside

http://synergy.cs.ucr.edu/

Thomas Repantis
Synergy: Quality of Service Support for Distributed Stream Processing Systems

Thomas Repantis
trep@cs.ucr.edu
http://www.cs.ucr.edu/~trep/

http://synergy.cs.ucr.edu/