

# **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](mailto:trep@cs.ucr.edu)

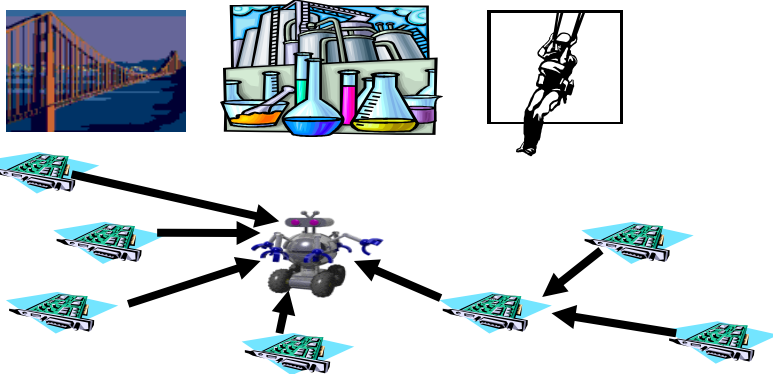
<http://www.cs.ucr.edu/~trep/>

# 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

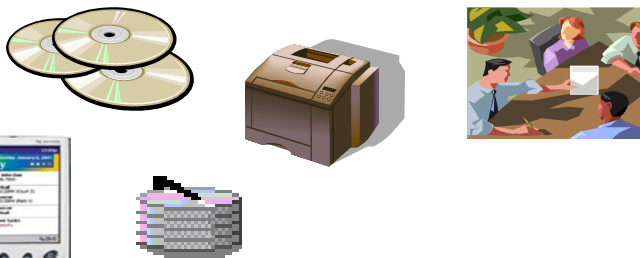
Network traffic monitoring for intrusion detection



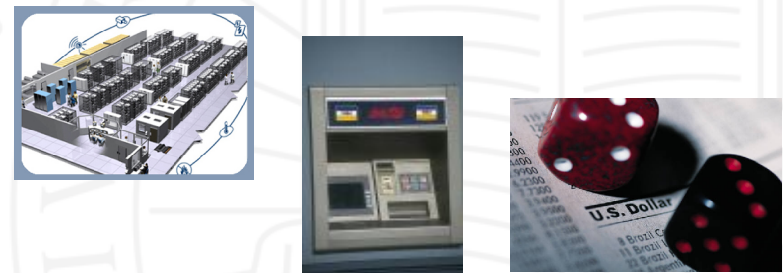
Analysis of readings coming from sensors or mobile robots



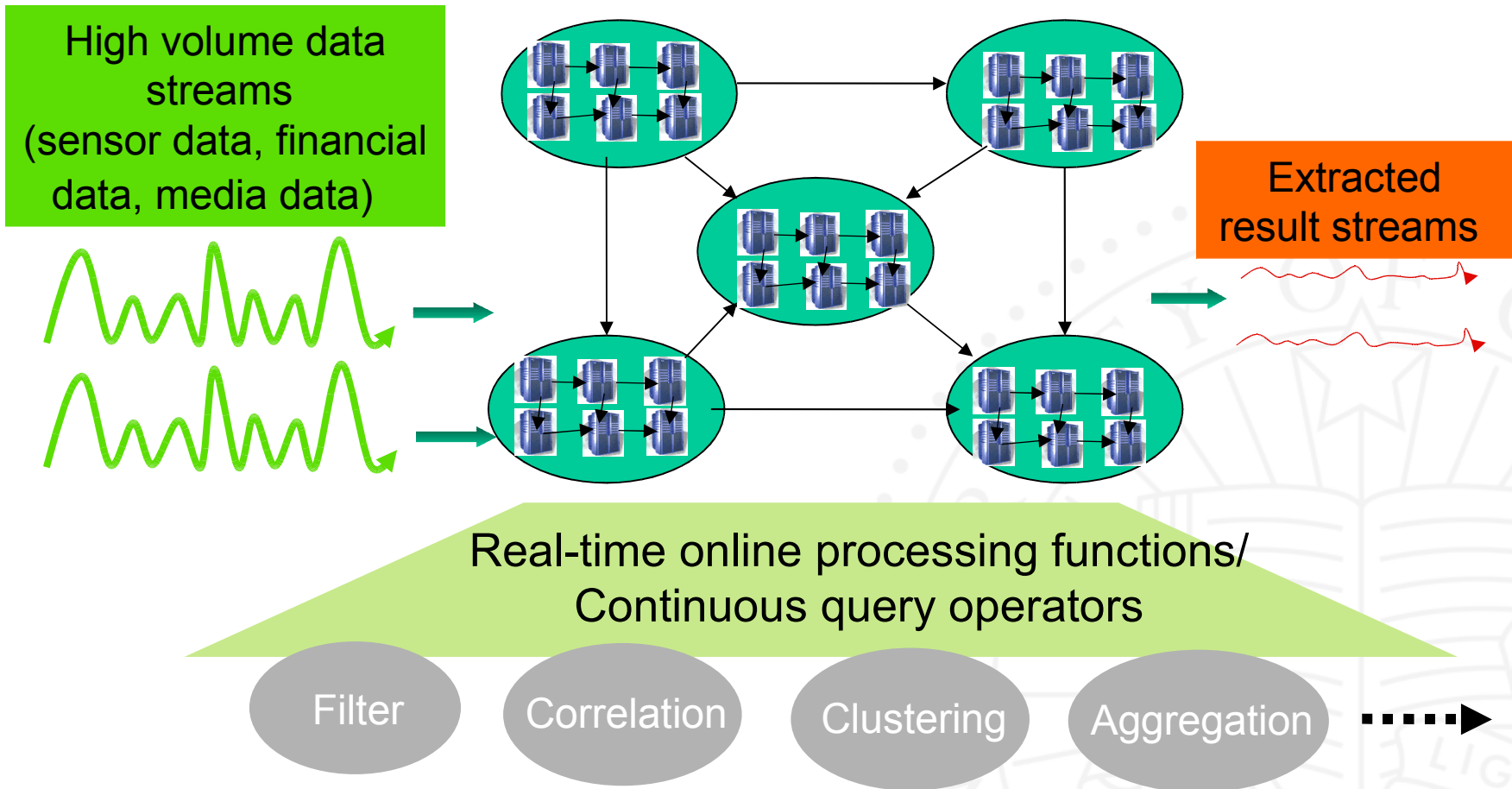
Click stream analysis for purchase recommendations or advertisements



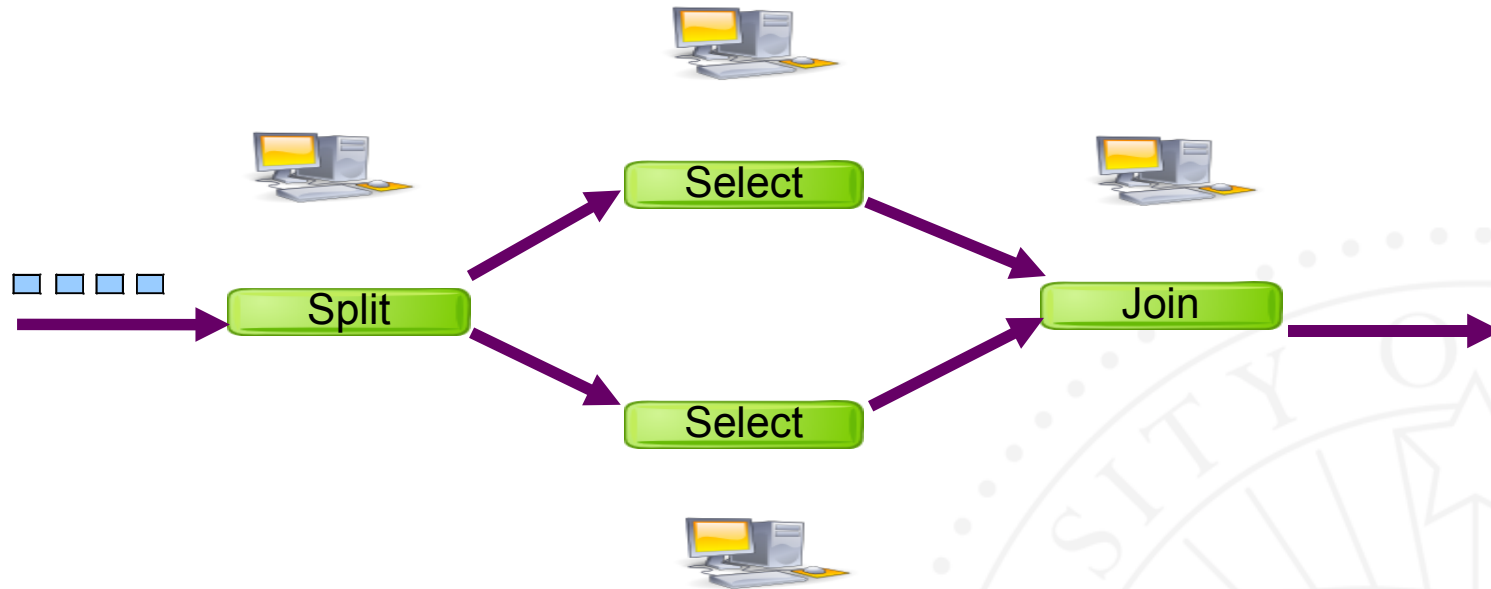
Customization of multimedia or news feeds



# Distributed Stream Processing System



# Stream Processing Environment



- 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

# QoS for Distributed Stream Processing Applications

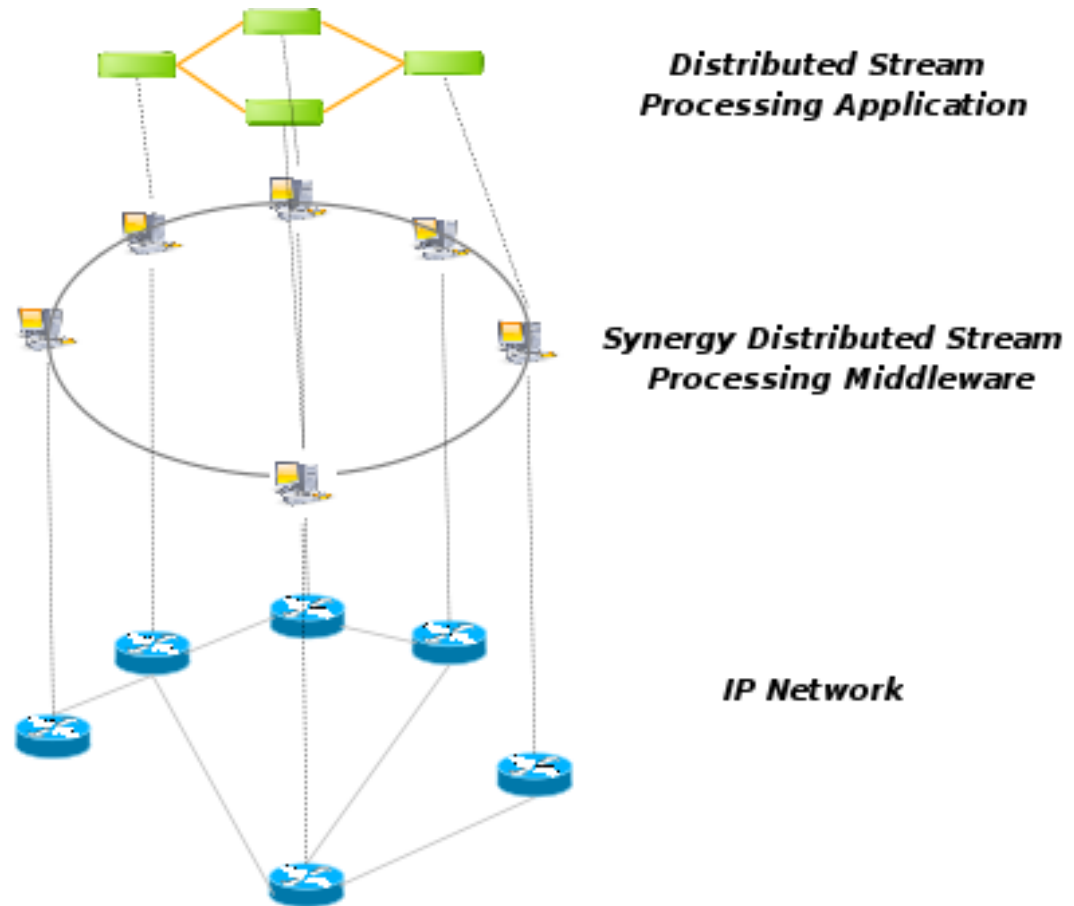
- **Our goal: How to run stream processing applications with QoS requirements, while efficiently managing system resources**
  - Share existing result streams
  - Share existing stream processing components
  - Predict QoS violations
  - Alleviate hot-spots
  - Maximize availability
- **Benefits**
  - Enhanced QoS provision
  - Reduced resource load
- **Challenges**
  - Concurrent component sharing
  - Highly dynamic environment
  - On-demand stream application requests
  - Scale that dictates decentralization

# 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

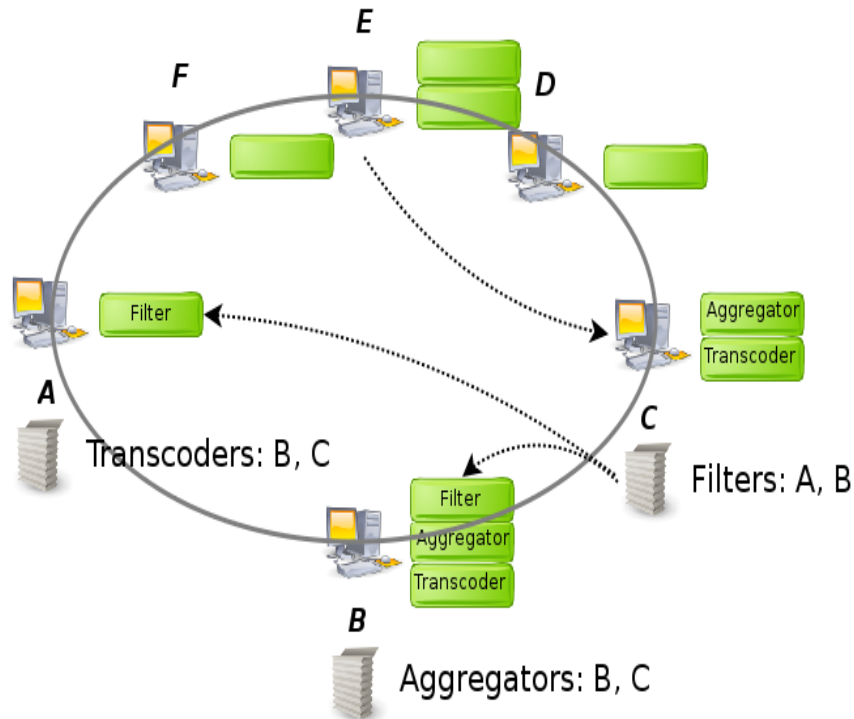
# Synergy Middleware

- A middleware managing the mappings:
  - From application layer to stream processing overlay layer
  - From stream processing overlay layer to physical resource layer



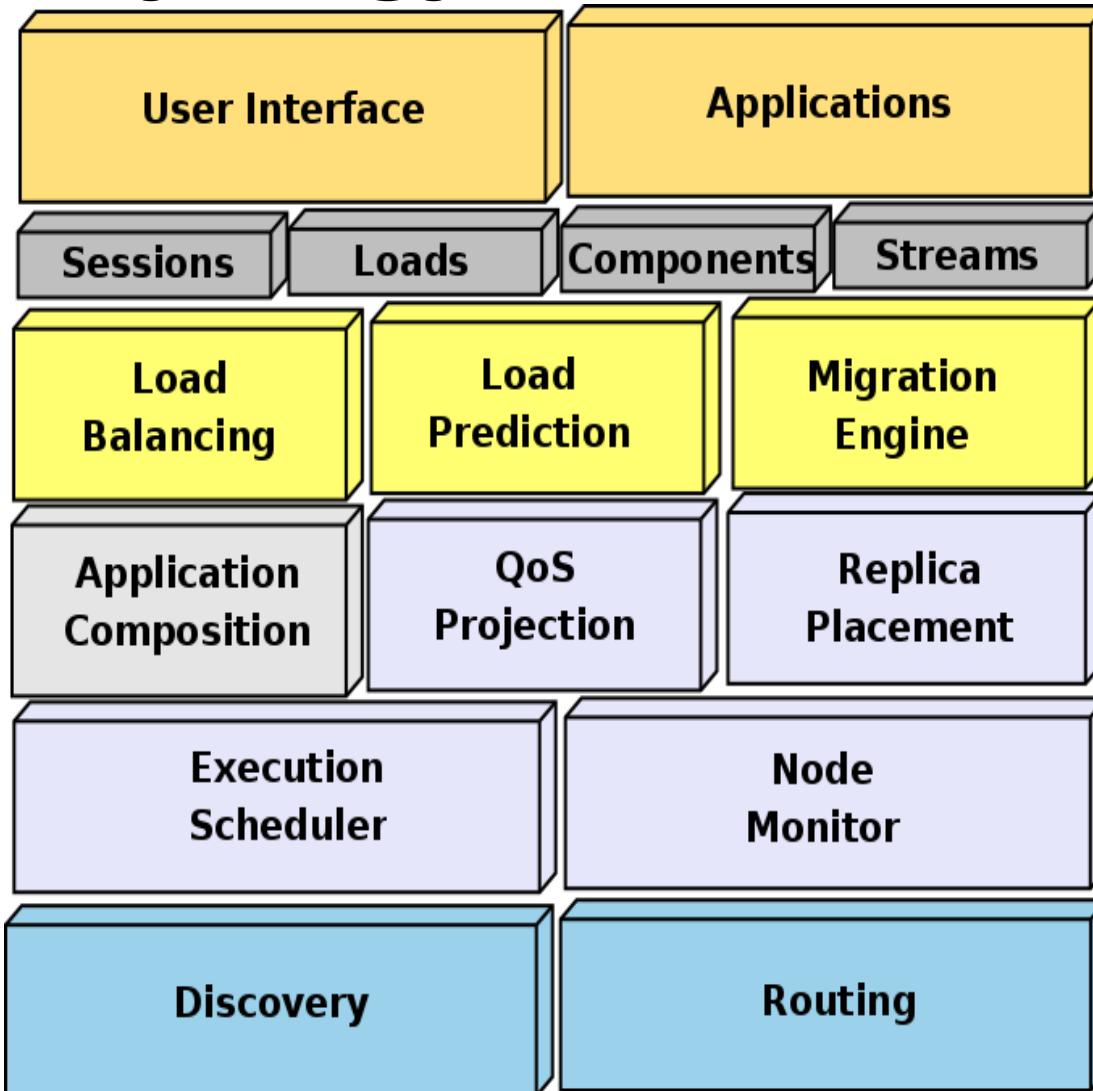


# Metadata Layer Over a DHT



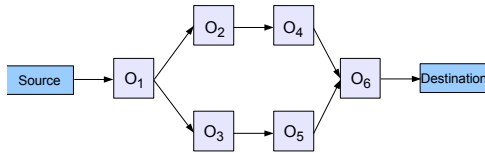
- Decouples stream and component placement from their discovery
- Stream and component names are hashed in a DHT
- DHT maps the hashed names to nodes currently offering the specified stream or component

# 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

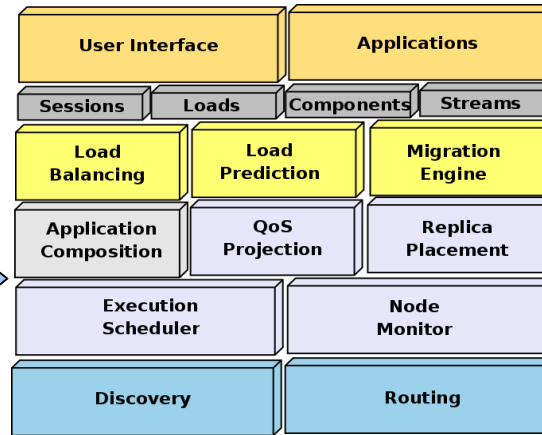
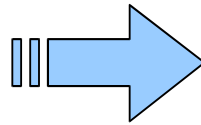
# Component Composition



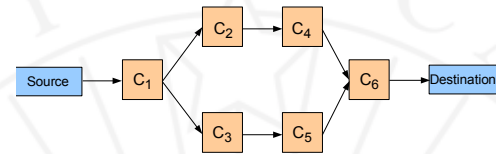
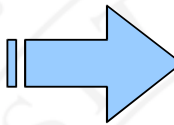
Query Plan

+

QoS Requirements

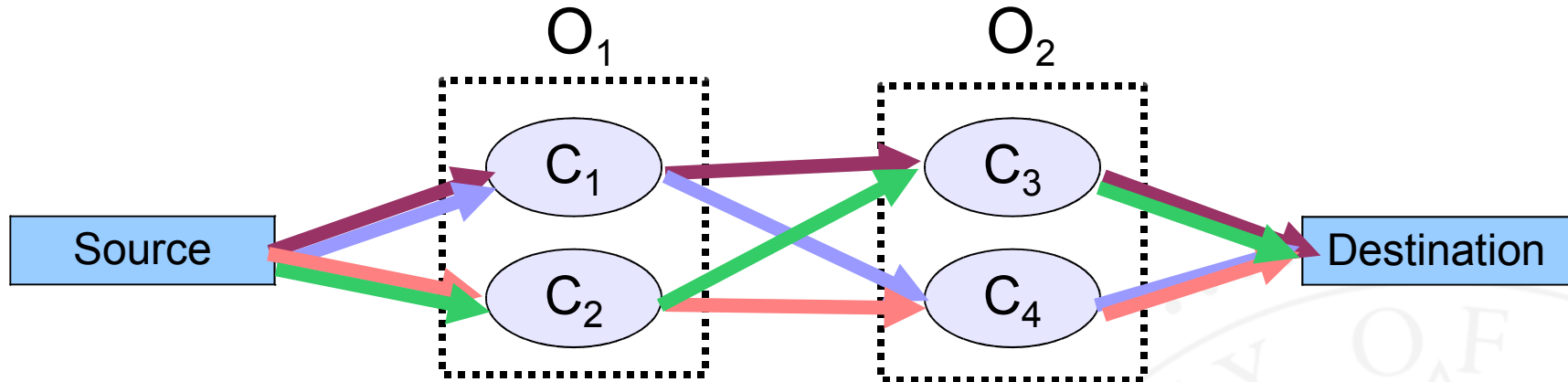


Synergy  
Middleware



Application  
Component  
Graph

# Composition Probes



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

# Composition Protocol

## Input

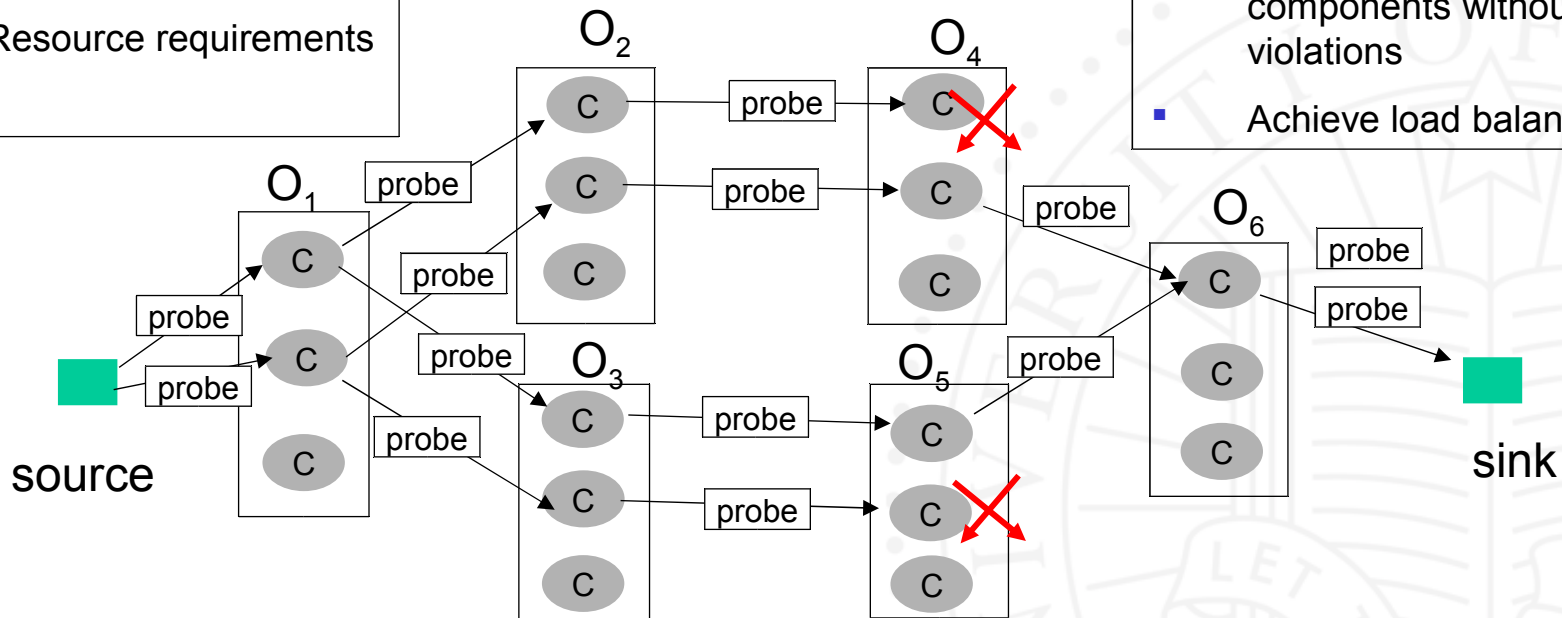
### Query Plan

- 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

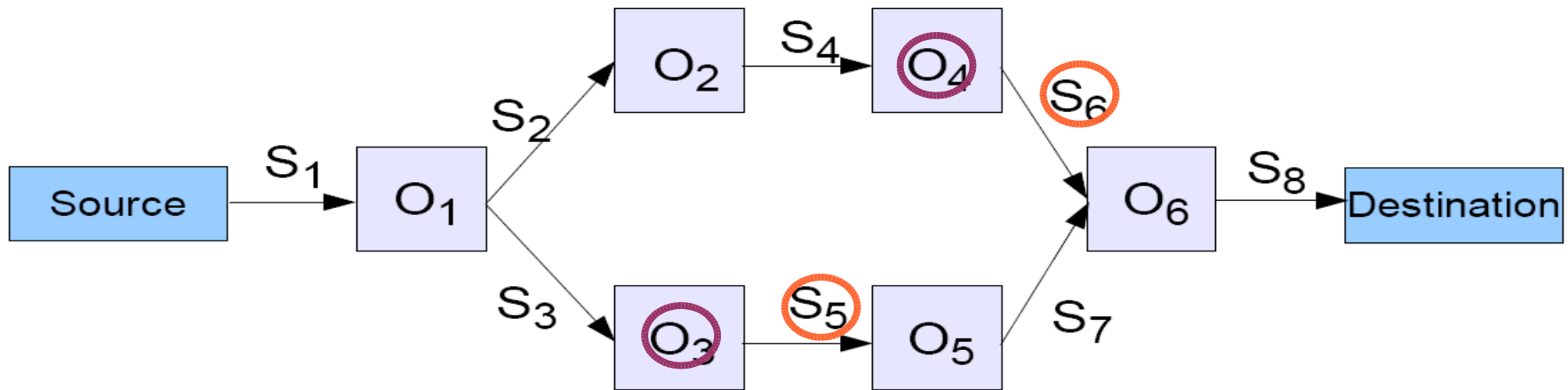
- All existing and the new application should not exceed requested execution time:

$$\delta + \hat{t} \leq q_t$$

- 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

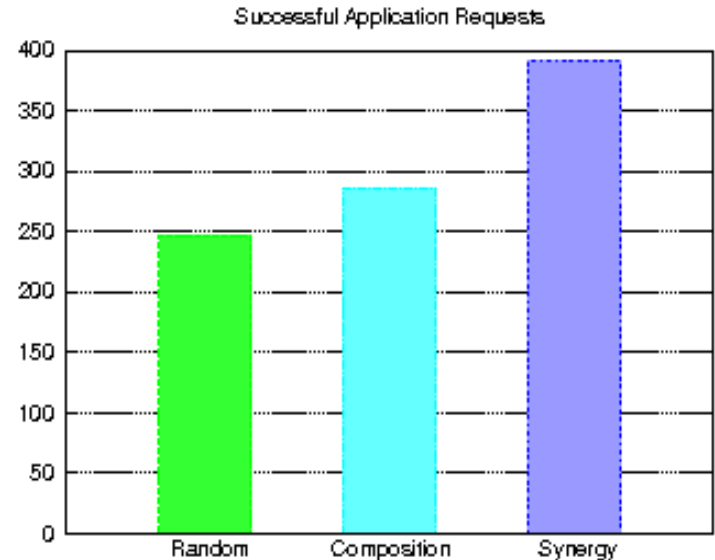
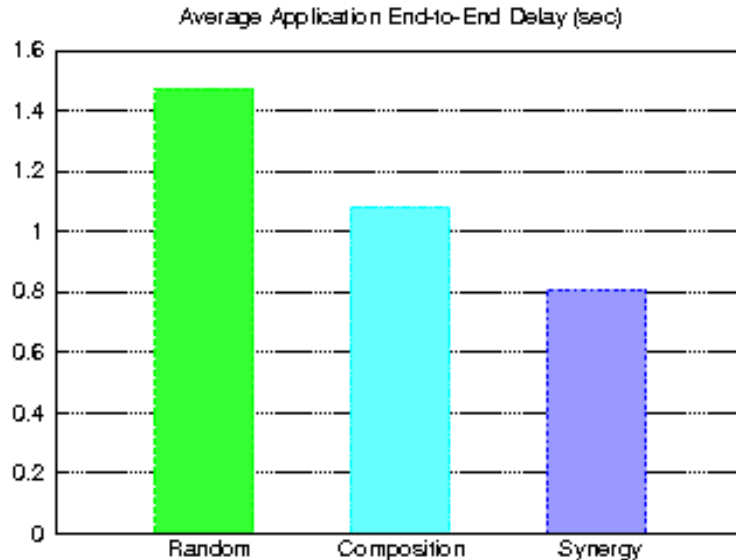
- Breadth first search on query plan to identify latest possible existing output streams
- Backtracking hop-by-hop, querying the metadata layer



# Experimental Setup

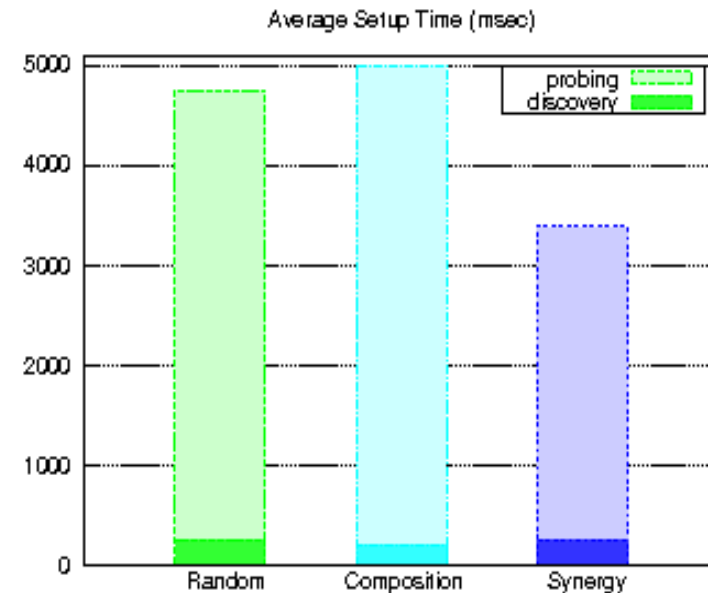
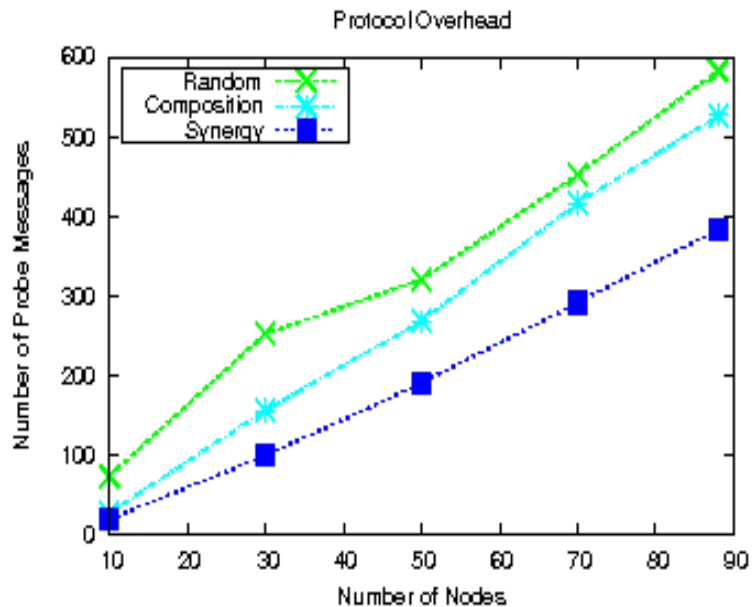
- ▶ PlanetLab multi-threaded prototype of about 35000 lines of Java running on 88 PlanetLab nodes
- ▶ Simulator of about 8500 lines of C++ for 500 random nodes of a GT-ITM topology of 1500 routers
- ▶ 5 replicas of each component
- ▶ Synergy vs Random, Greedy, and Composition

# Composition Performance



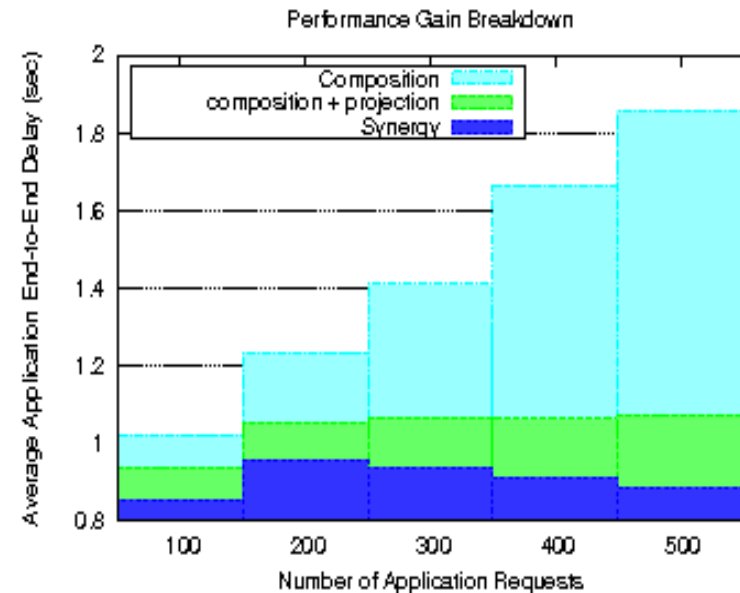
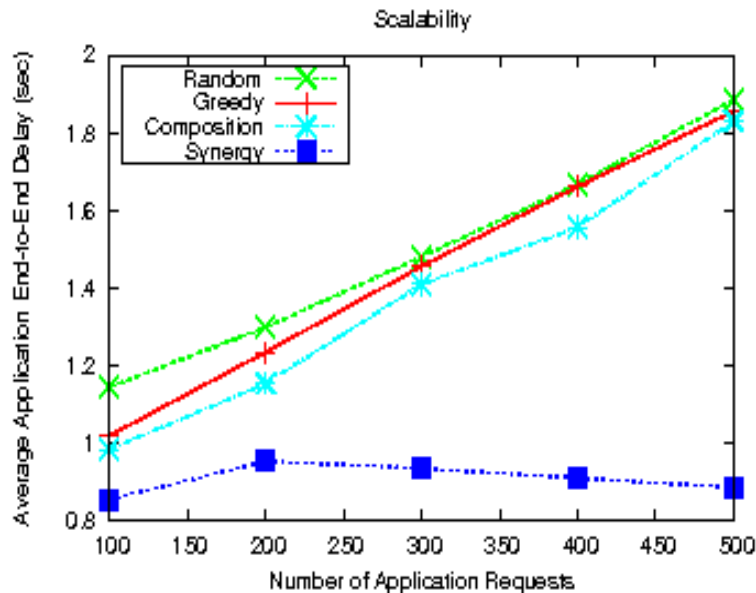
Stream reuse improves end-to-end delay by saving processing time and increases system capacity

# Composition Overhead



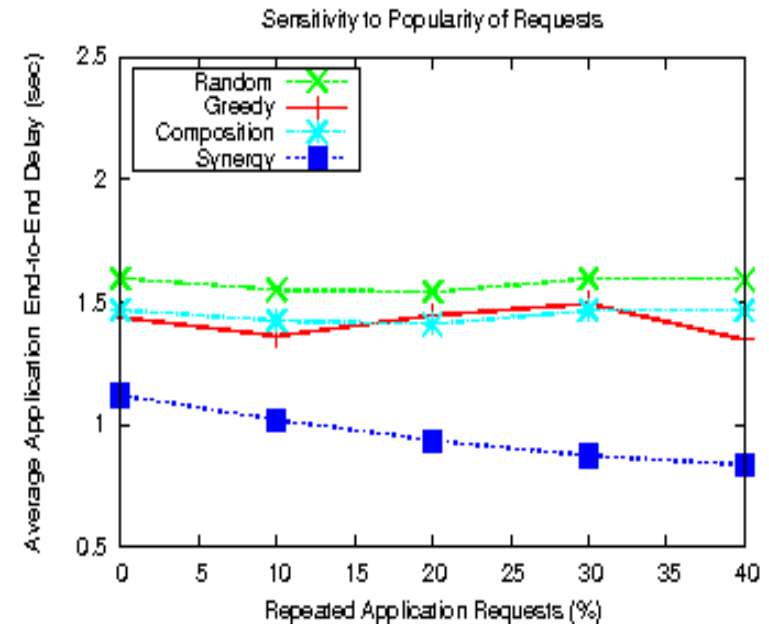
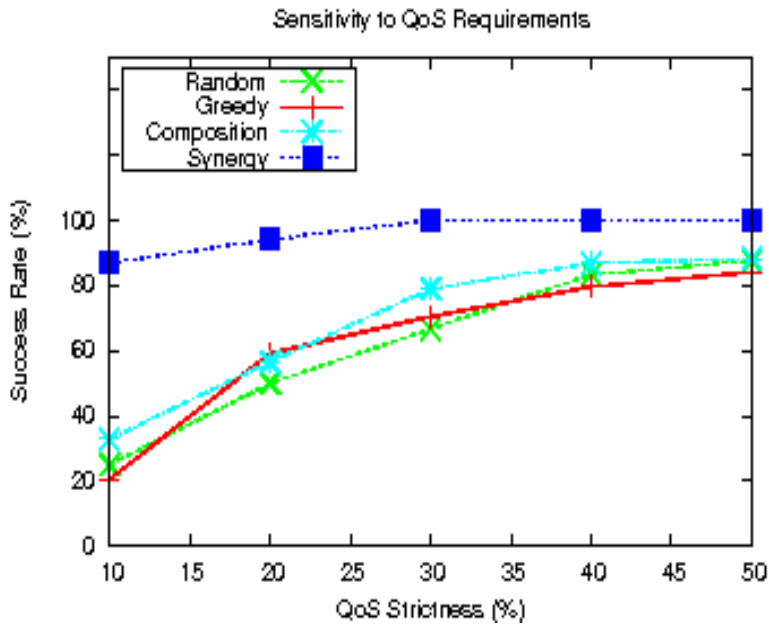
Stream reuse decreases probing overhead and setup time

# Performance on Simulator



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

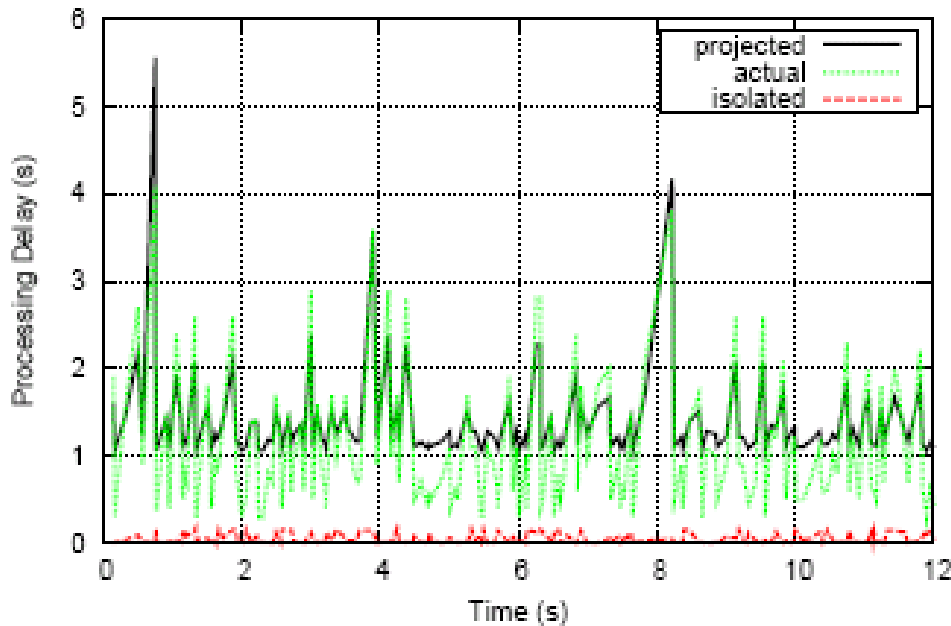
# Sensitivity on Simulator



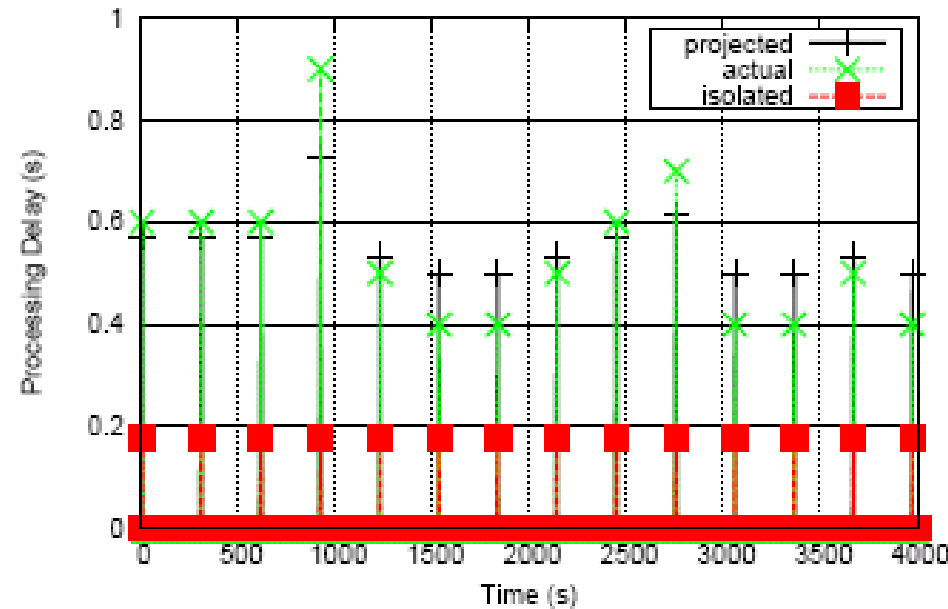
Synergy performs consistently better, regardless of QoS strictness or query popularity

# Projection Accuracy

Projection Accuracy (Network Traffic)



Projection Accuracy (Sensors Traffic)



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

# Roadmap

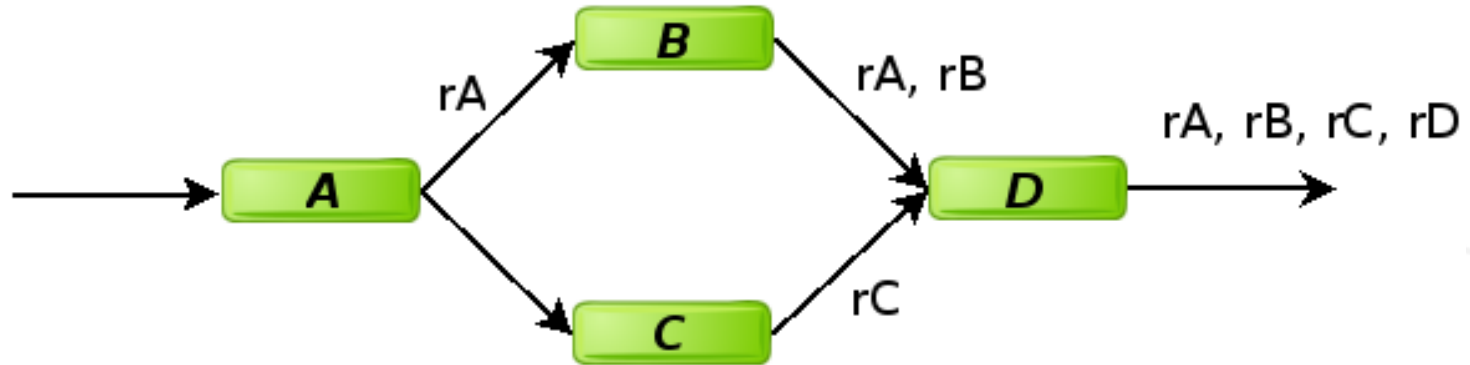
- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Component Composition
    - Composition Protocol
    - Component and Stream Sharing
  - Load Balancing
    - Hot-Spot Prediction
    - Hot-Spot Alleviation
  - High Availability
    - Replica Placement
- Conclusion
- Demo

# 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  $t_c$

$$t_{s(i)} = qt - \left( \sum_{j \in 1 \dots i-1} t_{c(j \rightarrow j+1)} + \sum_{j \in 1 \dots i-1} t_{e(j)} + \sum_{i \dots v-1} t_{c(j \rightarrow j+1)} + \sum_{j \in i \dots v} \hat{t}_{e(j)} \right) > 0$$

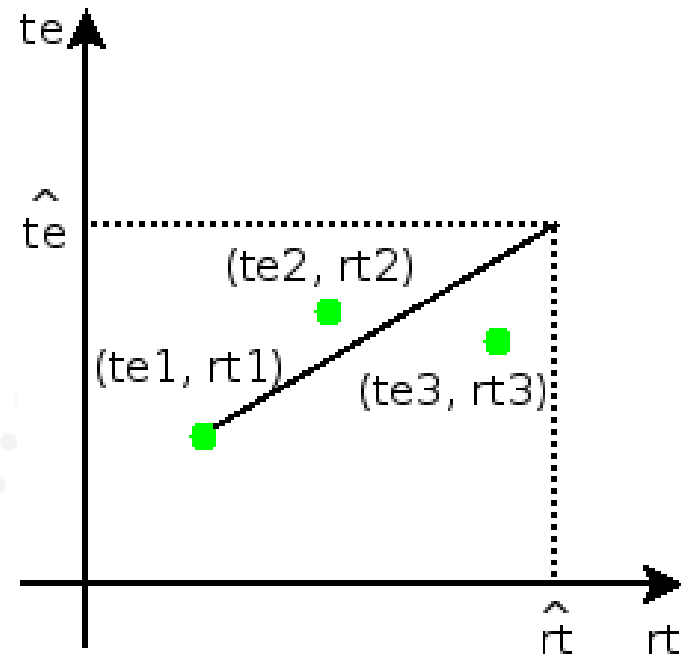
# Execution Time Prediction

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

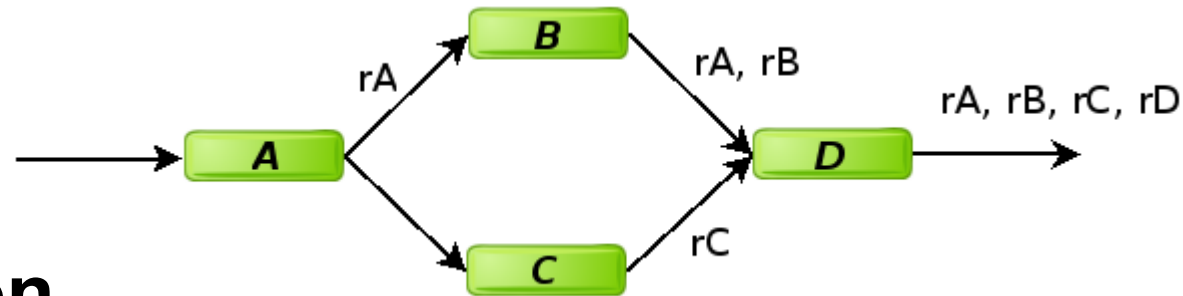
$$\hat{t}_e = a + b \cdot \hat{r}_t$$

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

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



# Rate Prediction



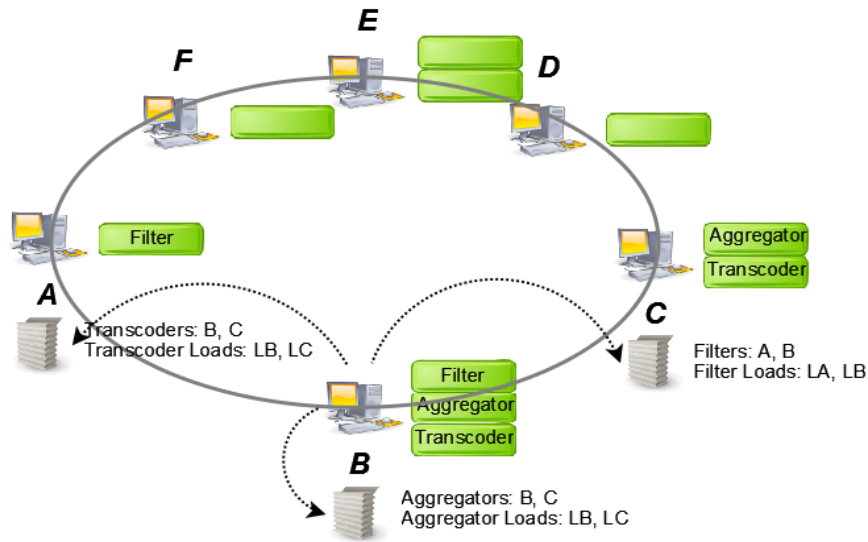
## ➤ Auto-correlation

$$\hat{r}_k = \frac{\arg_m \max R_{(k)}}{\arg_m \max R_{(k-1)}} \cdot r_{k-1} = \frac{r_{k(m)}}{r_{k-1(m)}} \cdot r_{k-1}$$

## ➤ Cross-correlation (Pearson Product Moment)

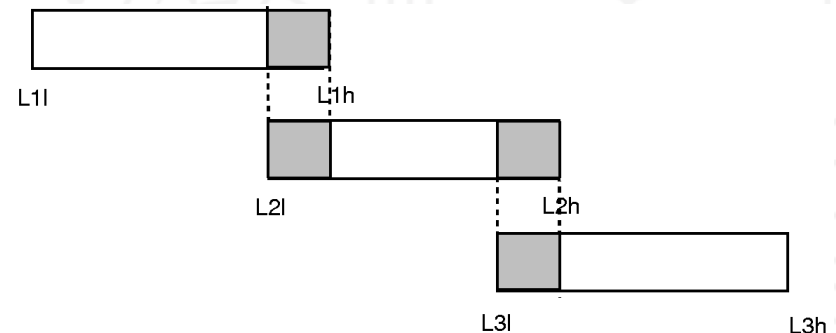
$$R_i = \frac{\sum_{j \in 1 \dots (k-1)} (r_{j(i)} - \bar{r}(i))(r_j - \bar{r})}{\sqrt{\sum_{j \in 1 \dots (k-1)} (r_{j(i)} - \bar{r}(i))^2 \sum_{j \in 1 \dots (k-1)} (r_j - \bar{r})^2}}$$

# Decentralized Load Monitoring

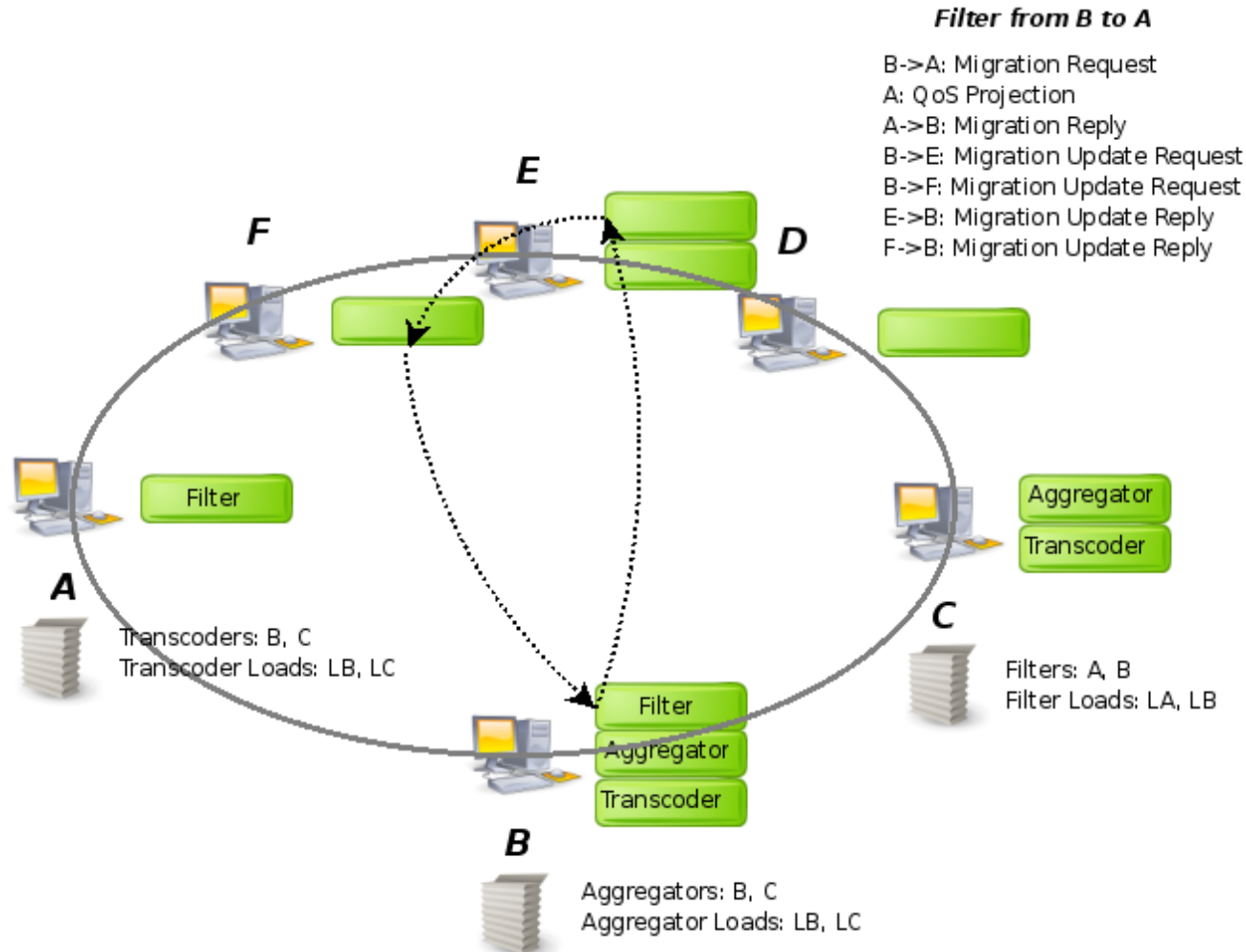


- DHT maps component names to the loads of peers hosting them
- Peers detect overloads and imbalances between all hosts of a component

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

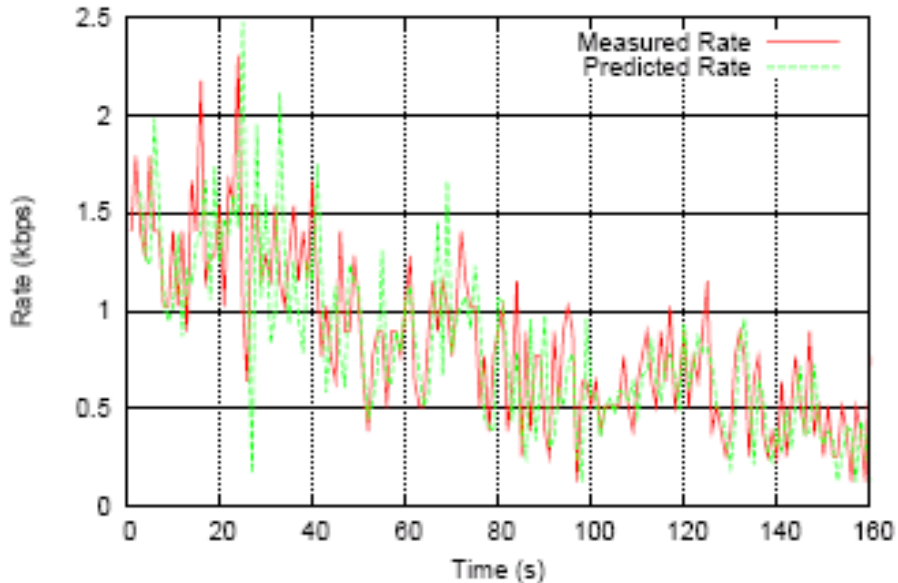


# Alleviating Hot-Spots via Migration

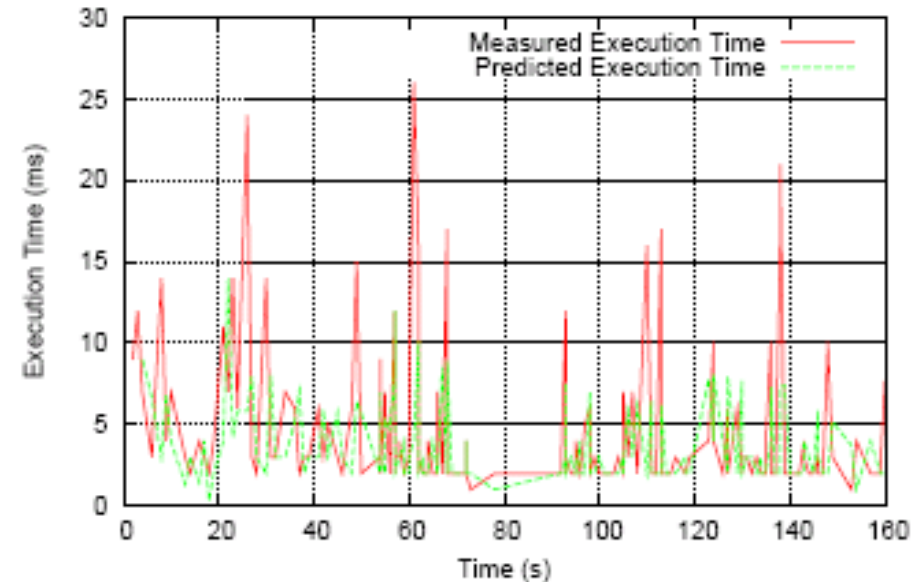


# Hot-Spot Prediction and Alleviation

Predicted vs Measured Rate for Compare



Predicted vs Measured Execution Time for Compare

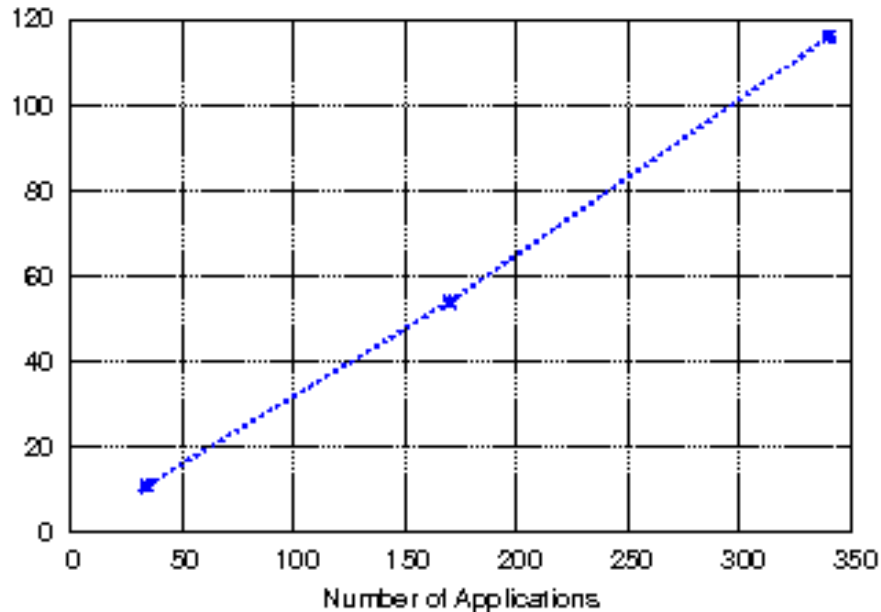


Average prediction error 3.7016%

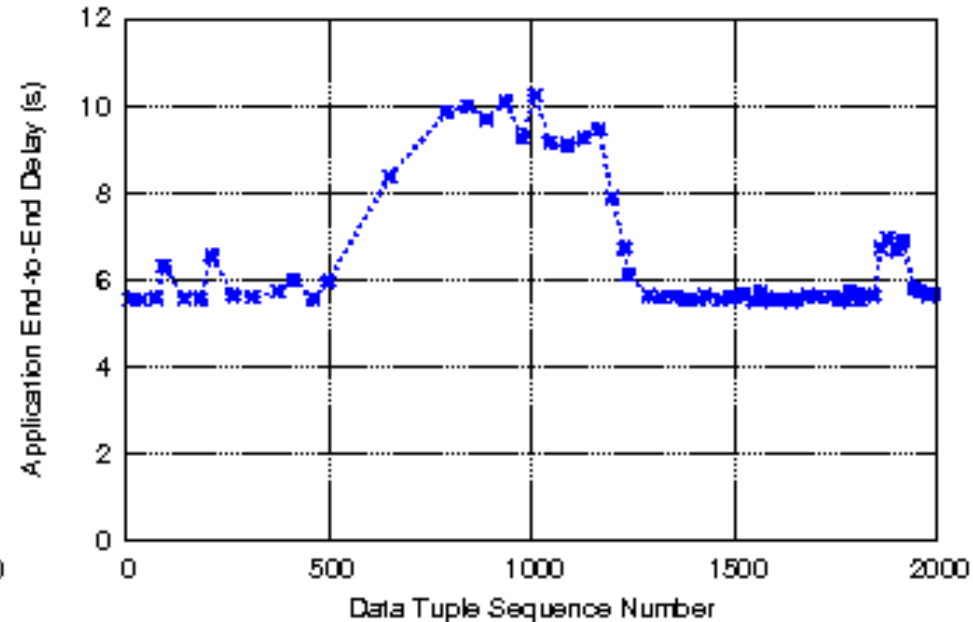
Average prediction overhead 0.5984ms

# Hot-Spot Prediction and Alleviation

Migration Overhead

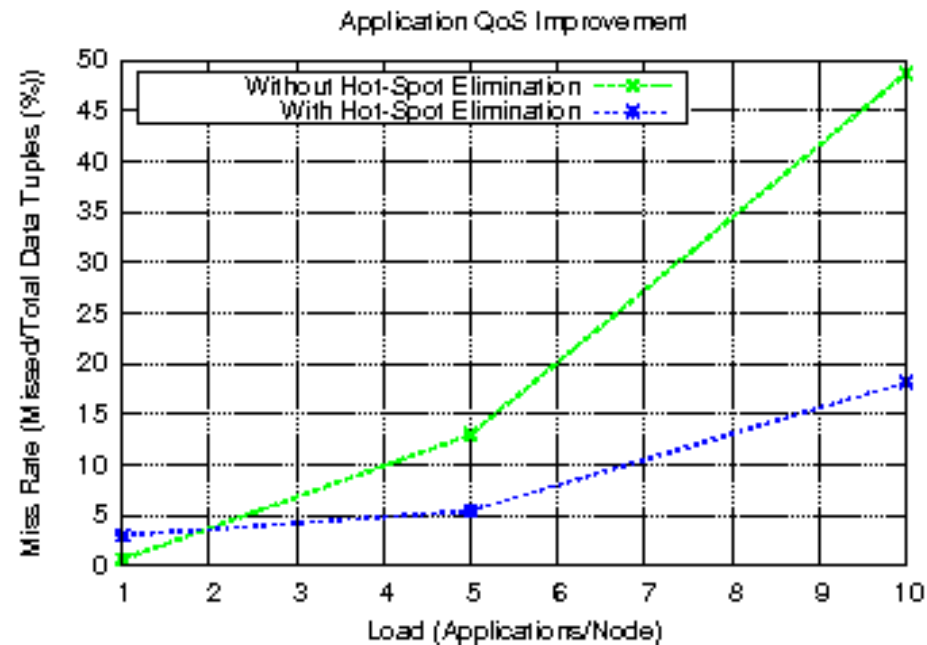
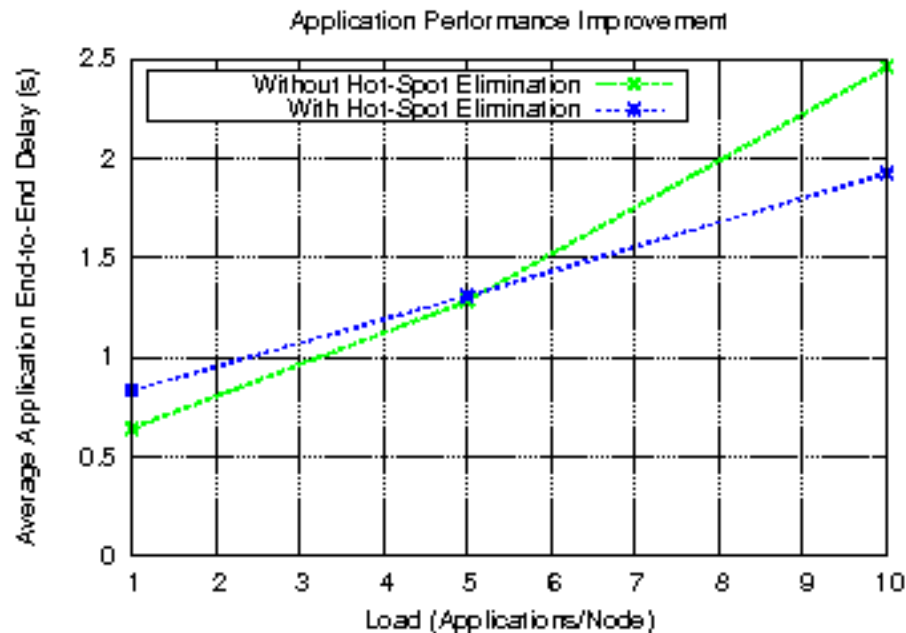


Application Performance Variation



Average one migration every three applications  
Average migration time 1144ms

# QoS Improvement



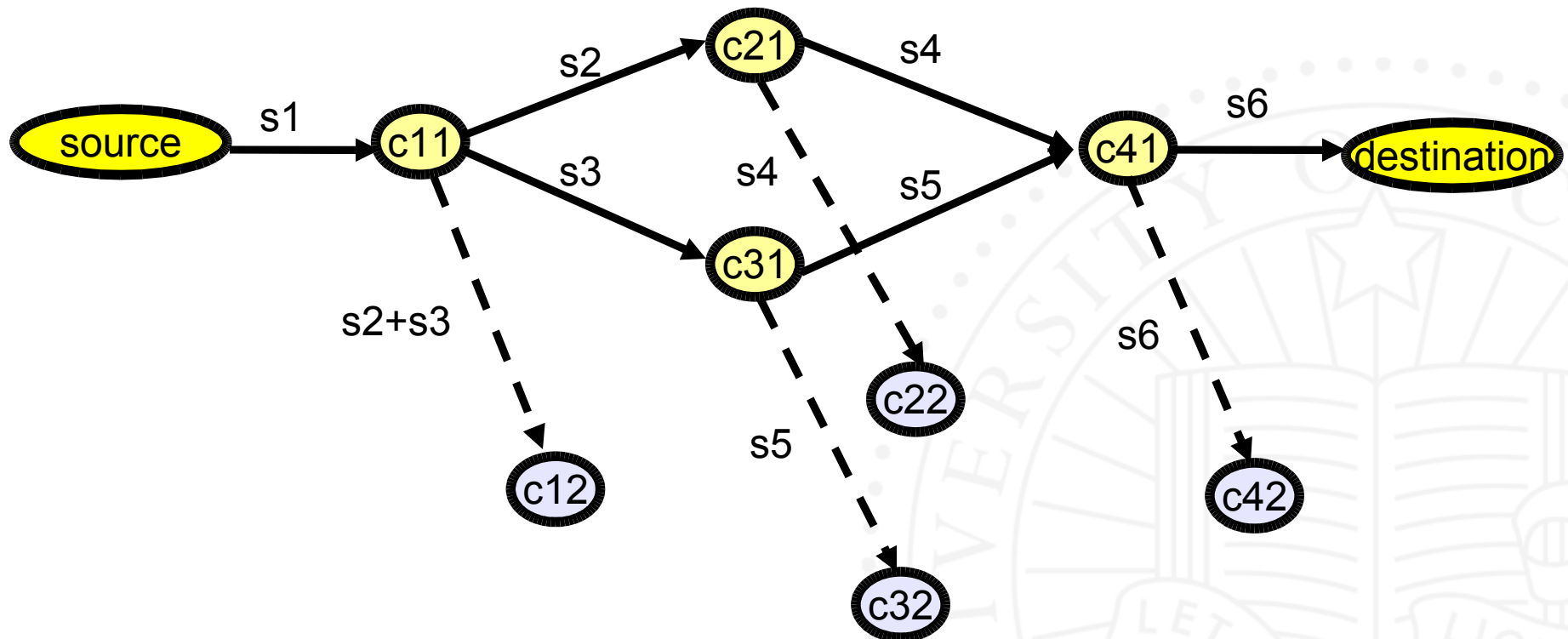
As load increases the benefits of hot-spot elimination become evident



# 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

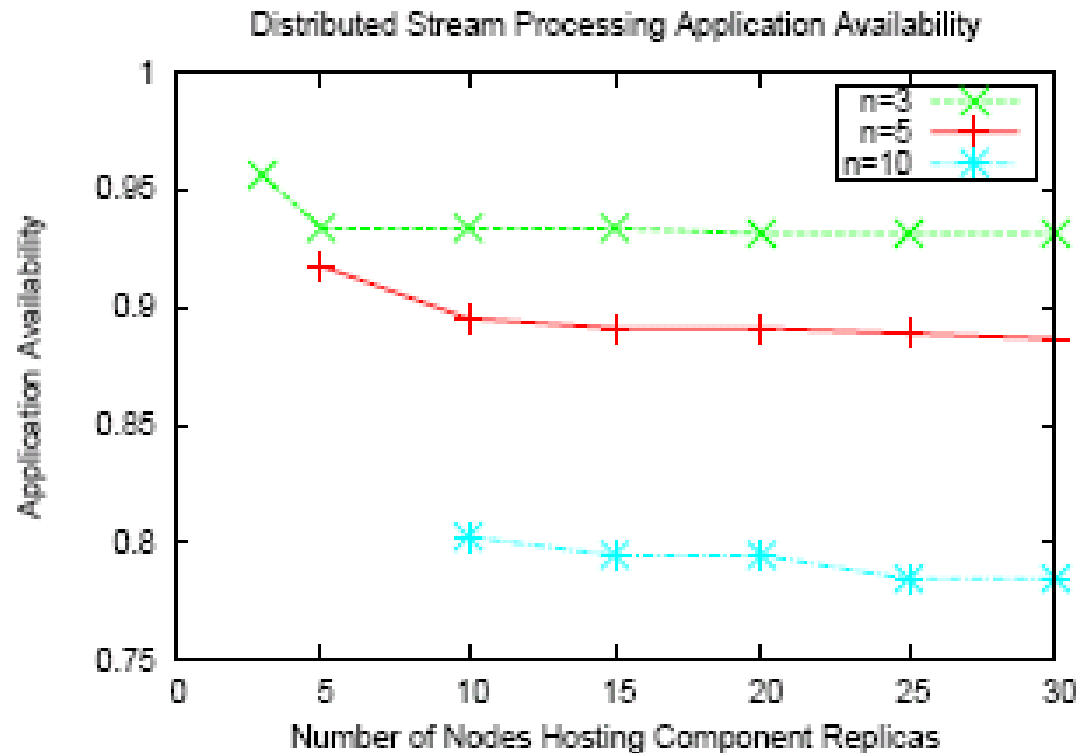
# Component Replication



# Component Replica Placement

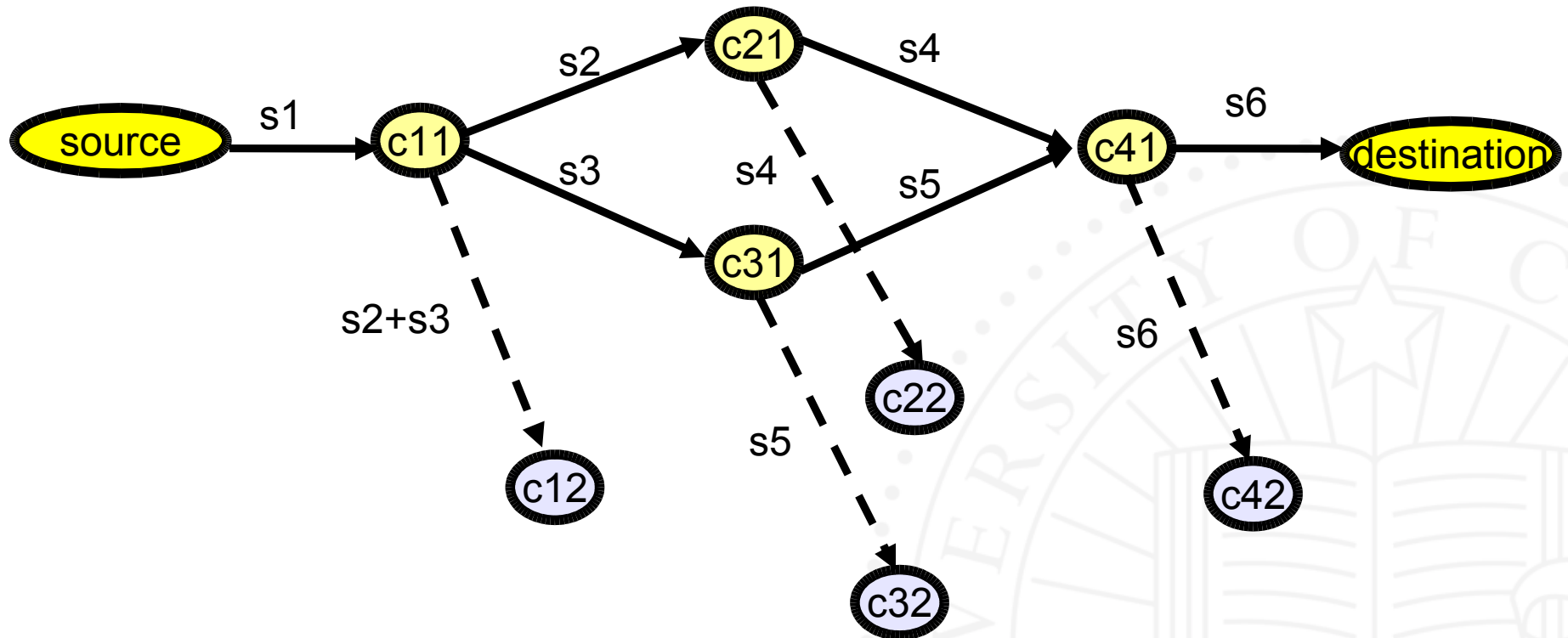
- Maximize availability of composite applications
  - Optimal: Place complete graph on each node
- Respect node resource availability
  - Processing capacity
  - Network bandwidth
- Maximize application performance
  - Inter-operator communication cost (between primaries)
  - Intra-operator communication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

# Placement for High Availability



Availability decreases with larger graphs and increases with higher concentration

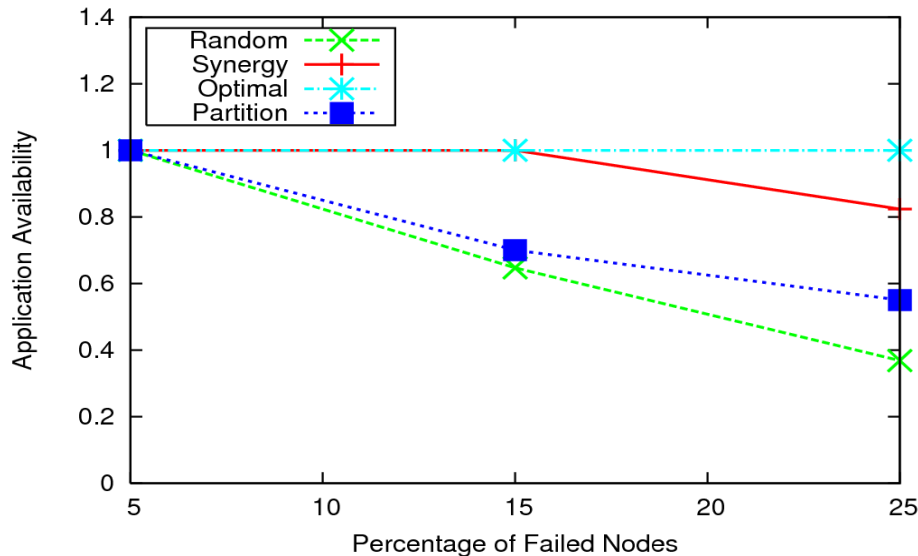
# Distributed Placement Protocol



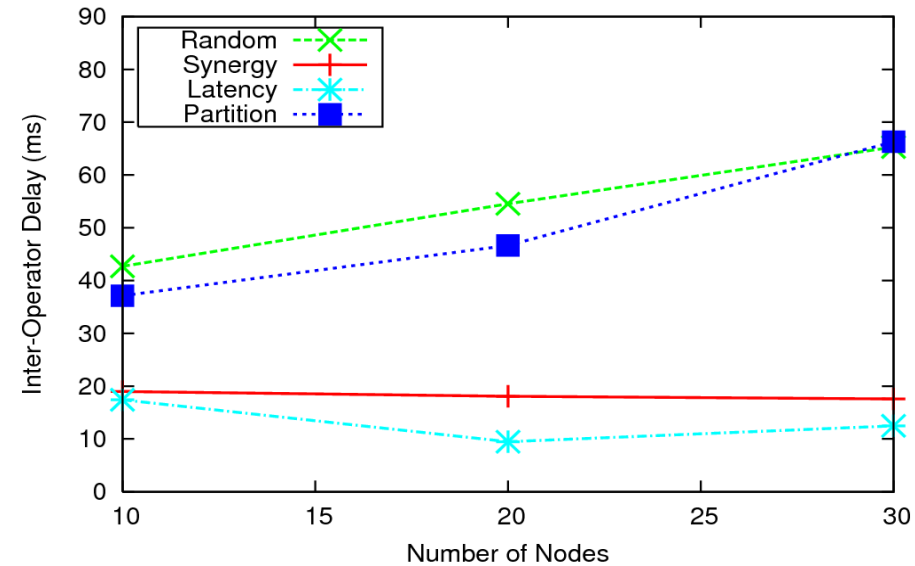
Closest used candidates

# Replica Placement

Effect of Failure Percentage on Availability



Effect of Scale on Inter-Operator Delay



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

# 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

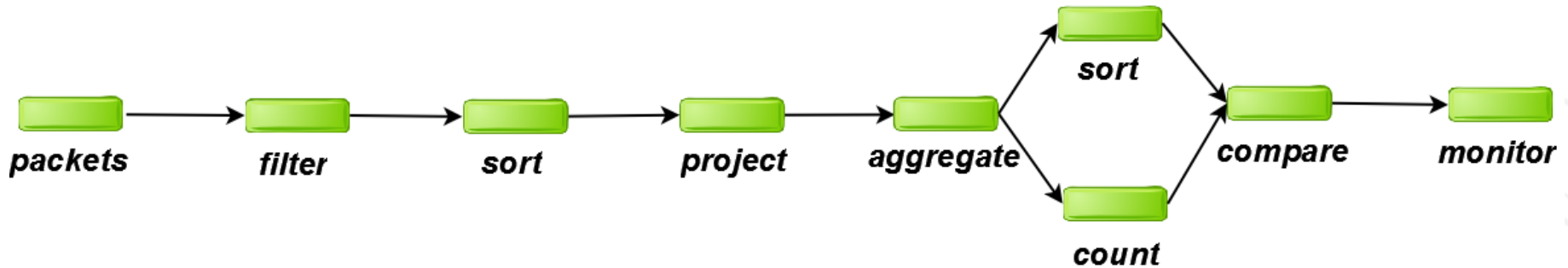
# Conclusion

- 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

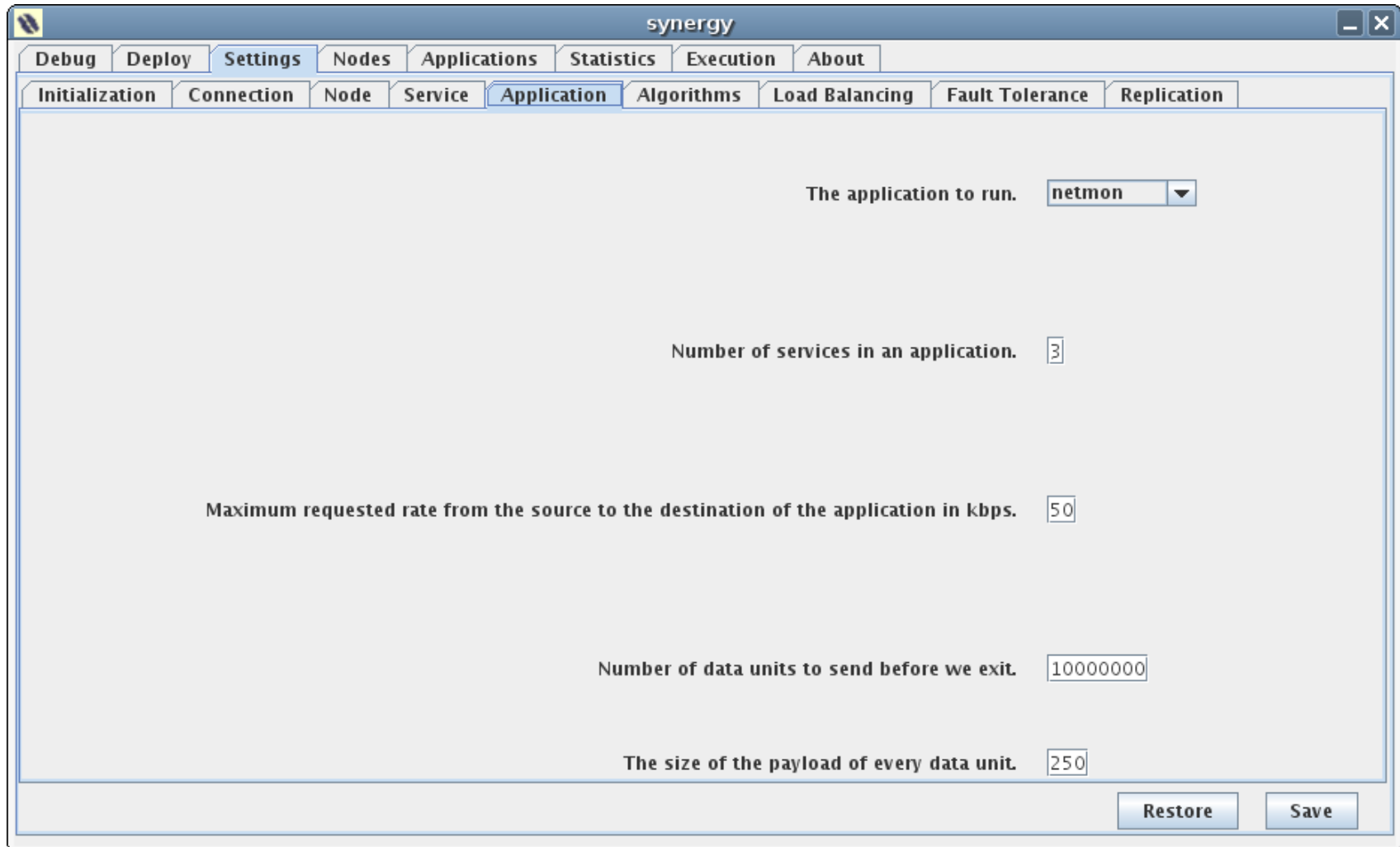
- Monitor source-destination pairs in top 5% of total traffic over last 20 minutes [Stream Query Repository]



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

|           |          |               |            |                 |      |
|-----------|----------|---------------|------------|-----------------|------|
| timestamp | sourceIP | destinationIP | sourcePort | destinationPort | size |
|-----------|----------|---------------|------------|-----------------|------|

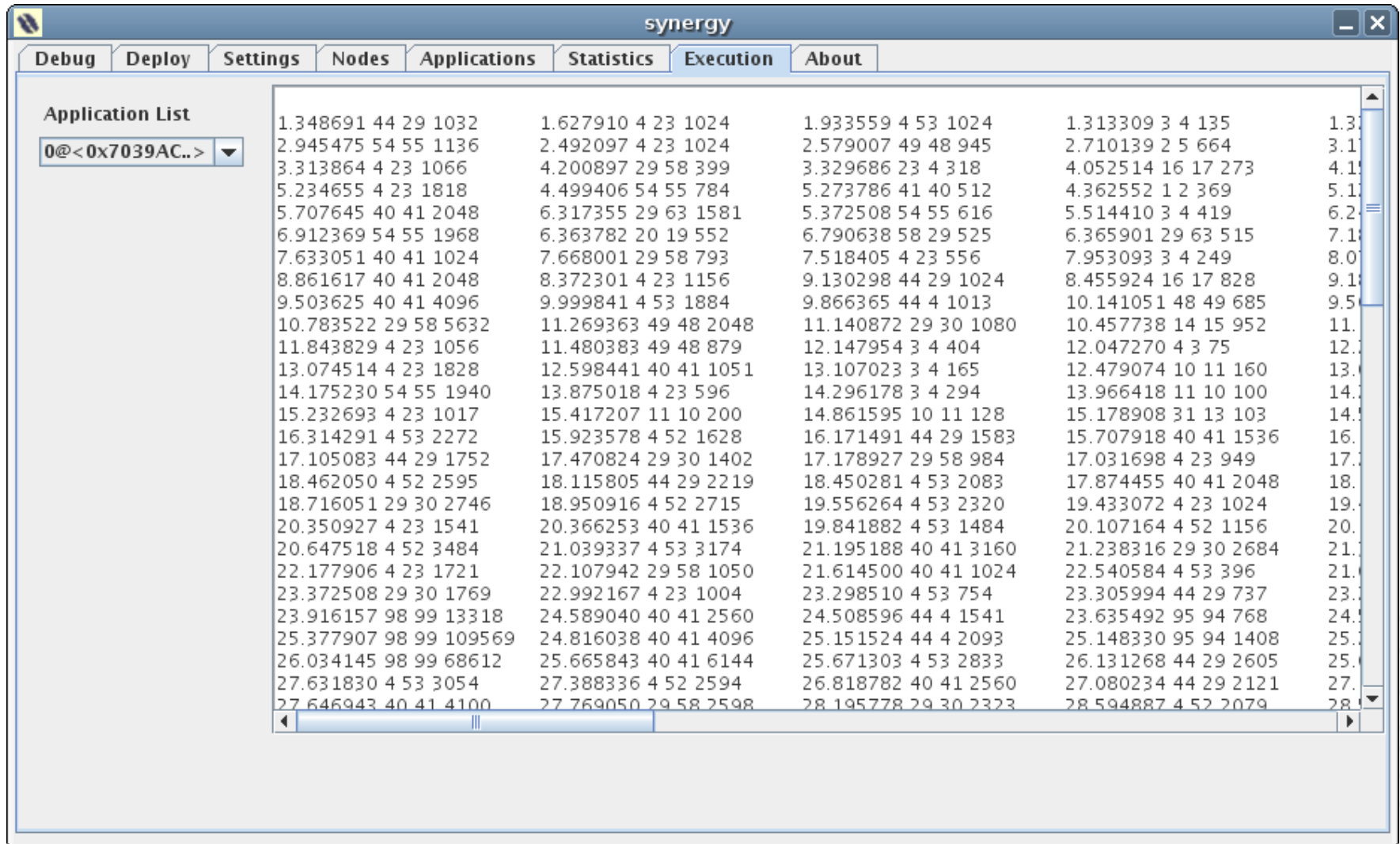
# GUI Settings



# GUI Application

The screenshot shows the synergy GUI interface. At the top, there are tabs for Debug, Deploy, Settings, Nodes, Applications, Statistics, Execution, and About. The main window displays a network diagram with three nodes: heartland:9050, heartland:9060, and heartland:9055. Components are represented by green boxes and connected by solid and dashed arrows. The components include DataSource, DataReceiver, Compare, Count, Aggregate, Project, SortIP, and FilterTime. A left sidebar contains controls for 'Change Application Color', 'Picking Mouse Mode', and checkboxes for 'Show Hierarchy', 'Show Components', and 'Show Nodes'. Below these are lists for 'Inactive components', 'Active components', and 'CPU load/Bandwidth load'.

# GUI Execution



The screenshot shows the synergy application window with the Execution tab selected. The window title is "synergy". The tabs are Debug, Deploy, Settings, Nodes, Applications, Statistics, Execution, and About. The Application List on the left shows "0@<0x7039AC..>". The main area displays a grid of numerical data in 5 columns and 28 rows.

|           |              |           |            |           |            |           |            |           |
|-----------|--------------|-----------|------------|-----------|------------|-----------|------------|-----------|
| 1.348691  | 44 29 1032   | 1.627910  | 4 23 1024  | 1.933559  | 4 53 1024  | 1.313309  | 3 4 135    | 1.313309  |
| 2.945475  | 54 55 1136   | 2.492097  | 4 23 1024  | 2.579007  | 49 48 945  | 2.710139  | 2 5 664    | 3.10139   |
| 3.313864  | 4 23 1066    | 4.200897  | 29 58 399  | 3.329686  | 23 4 318   | 4.052514  | 16 17 273  | 4.105251  |
| 5.234655  | 4 23 1818    | 4.499406  | 54 55 784  | 5.273786  | 41 40 512  | 4.362552  | 1 2 369    | 5.105252  |
| 5.707645  | 40 41 2048   | 6.317355  | 29 63 1581 | 5.372508  | 54 55 616  | 5.514410  | 3 4 419    | 6.205253  |
| 6.912369  | 54 55 1968   | 6.363782  | 20 19 552  | 6.790638  | 58 29 525  | 6.365901  | 29 63 515  | 7.105254  |
| 7.633051  | 40 41 1024   | 7.668001  | 29 58 793  | 7.518405  | 4 23 556   | 7.953093  | 3 4 249    | 8.005255  |
| 8.861617  | 40 41 2048   | 8.372301  | 4 23 1156  | 9.130298  | 44 29 1024 | 8.455924  | 16 17 828  | 9.105256  |
| 9.503625  | 40 41 4096   | 9.999841  | 4 53 1884  | 9.866365  | 44 4 1013  | 10.141051 | 48 49 685  | 9.505257  |
| 10.783522 | 29 58 5632   | 11.269363 | 49 48 2048 | 11.140872 | 29 30 1080 | 10.457738 | 14 15 952  | 11.105258 |
| 11.843829 | 4 23 1056    | 11.480383 | 49 48 879  | 12.147954 | 3 4 404    | 12.047270 | 4 3 75     | 12.105259 |
| 13.074514 | 4 23 1828    | 12.598441 | 40 41 1051 | 13.107023 | 3 4 165    | 12.479074 | 10 11 160  | 13.105260 |
| 14.175230 | 54 55 1940   | 13.875018 | 4 23 596   | 14.296178 | 3 4 294    | 13.966418 | 11 10 100  | 14.105261 |
| 15.232693 | 4 23 1017    | 15.417207 | 11 10 200  | 14.861595 | 10 11 128  | 15.178908 | 31 13 103  | 14.105262 |
| 16.314291 | 4 53 2272    | 15.923578 | 4 52 1628  | 16.171491 | 44 29 1583 | 15.707918 | 40 41 1536 | 16.105263 |
| 17.105083 | 44 29 1752   | 17.470824 | 29 30 1402 | 17.178927 | 29 58 984  | 17.031698 | 4 23 949   | 17.105264 |
| 18.462050 | 4 52 2595    | 18.115805 | 44 29 2219 | 18.450281 | 4 53 2083  | 17.874455 | 40 41 2048 | 18.105265 |
| 18.716051 | 29 30 2746   | 18.950916 | 4 52 2715  | 19.556264 | 4 53 2320  | 19.433072 | 4 23 1024  | 19.105266 |
| 20.350927 | 4 23 1541    | 20.366253 | 40 41 1536 | 19.841882 | 4 53 1484  | 20.107164 | 4 52 1156  | 20.105267 |
| 20.647518 | 4 52 3484    | 21.039337 | 4 53 3174  | 21.195188 | 40 41 3160 | 21.238316 | 29 30 2684 | 21.105268 |
| 22.177906 | 4 23 1721    | 22.107942 | 29 58 1050 | 22.614500 | 40 41 1024 | 22.540584 | 4 53 396   | 21.105269 |
| 23.372508 | 29 30 1769   | 22.992167 | 4 23 1004  | 23.298510 | 4 53 754   | 23.305994 | 44 29 737  | 23.105270 |
| 23.916157 | 98 99 13318  | 24.589040 | 40 41 2560 | 24.508596 | 44 4 1541  | 23.635492 | 95 94 768  | 24.105271 |
| 25.377907 | 98 99 109569 | 24.816038 | 40 41 4096 | 25.151524 | 44 4 2093  | 25.148330 | 95 94 1408 | 25.105272 |
| 26.034145 | 98 99 68612  | 25.665843 | 40 41 6144 | 25.671303 | 4 53 2833  | 26.131268 | 44 29 2605 | 25.105273 |
| 27.631830 | 4 53 3054    | 27.388336 | 4 52 2594  | 26.818782 | 40 41 2560 | 27.080234 | 44 29 2121 | 27.105274 |
| 27.646943 | 40 41 4100   | 27.769050 | 29 58 2598 | 28.195778 | 29 30 2323 | 28.594887 | 4 52 2079  | 28.105275 |

# 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/>

# Synergy: Quality of Service Support for Distributed Stream Processing Systems

*Thomas Repantis*

[trep@cs.ucr.edu](mailto:trep@cs.ucr.edu)

<http://www.cs.ucr.edu/~trep/>



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