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/

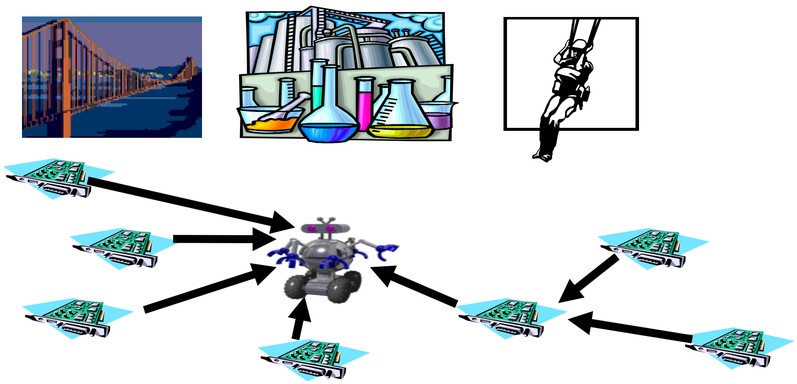
UCR

UNIVERSITY OF CALIFORNIA, RIVERSIDE

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]

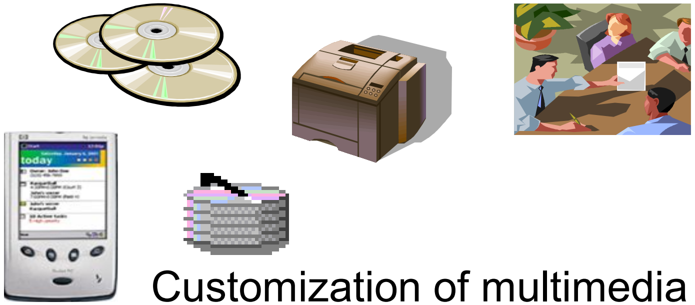
On-Line Data Stream Processing



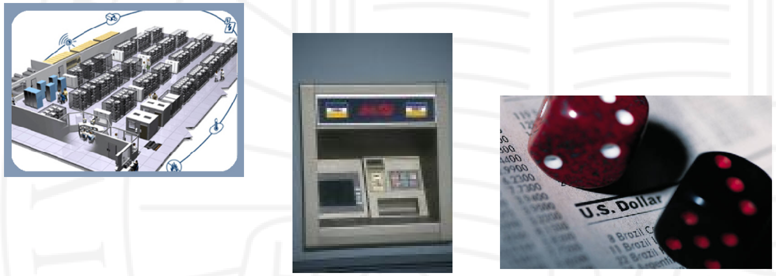
Analysis of readings coming from sensors or mobile robots

Network traffic monitoring for intrusion detection

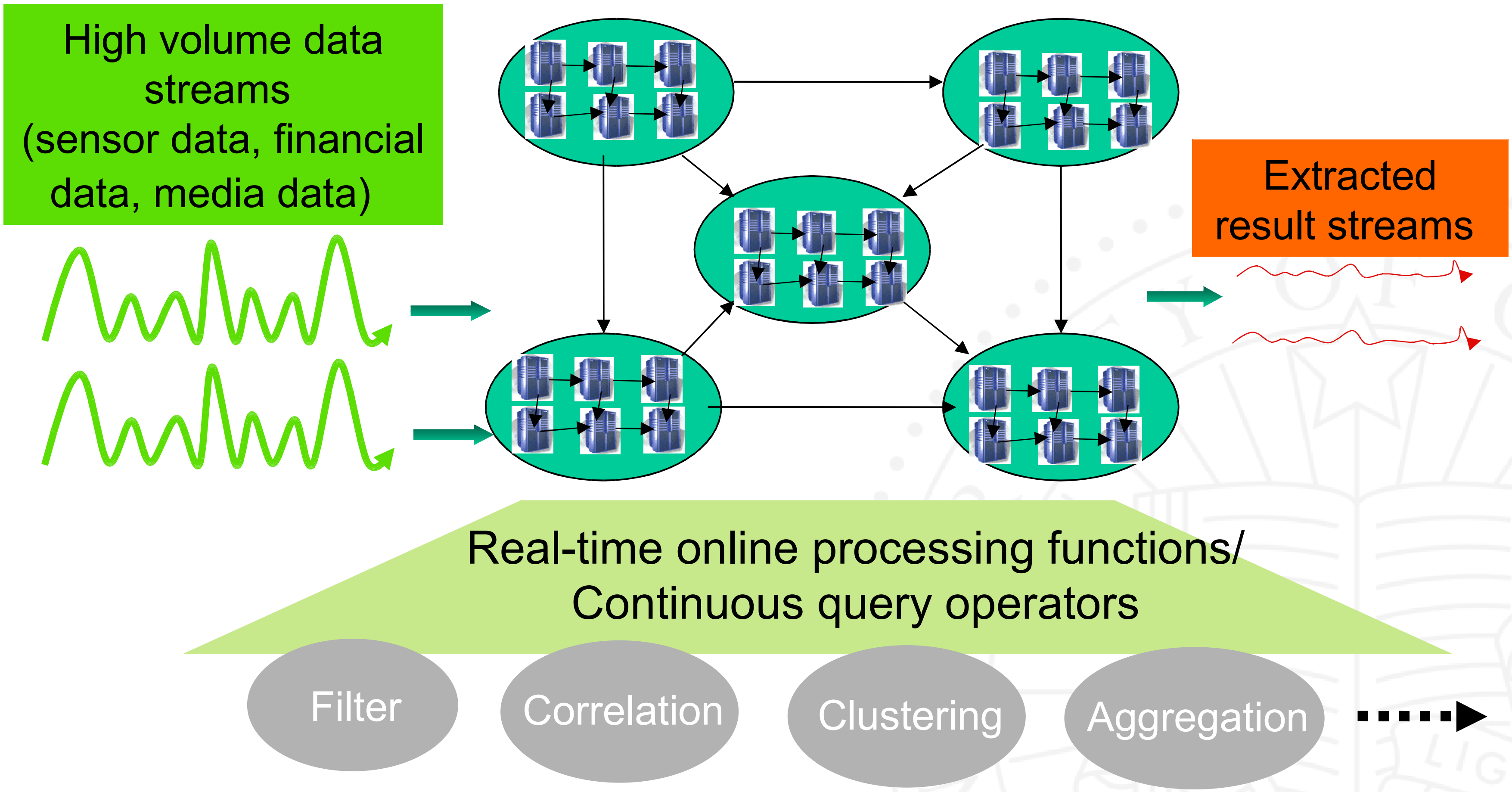
Click stream analysis for purchase recommendations or advertisements



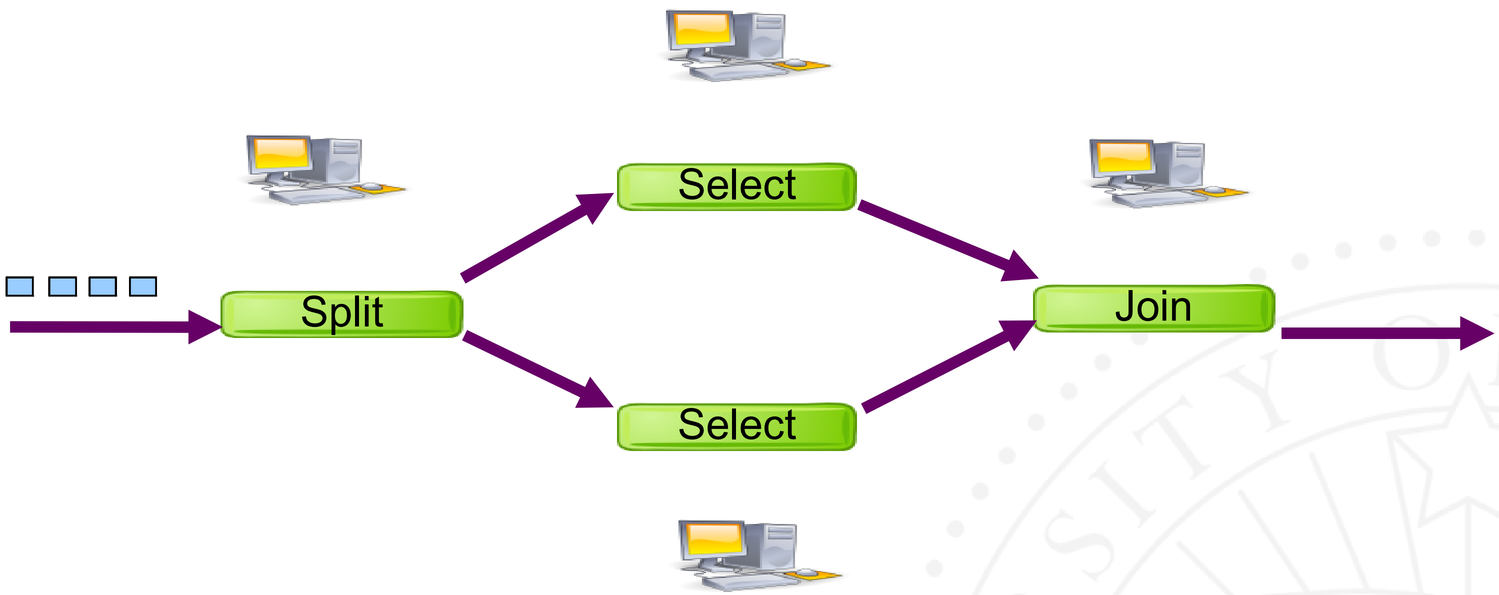
Customization of multimedia or news feeds



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

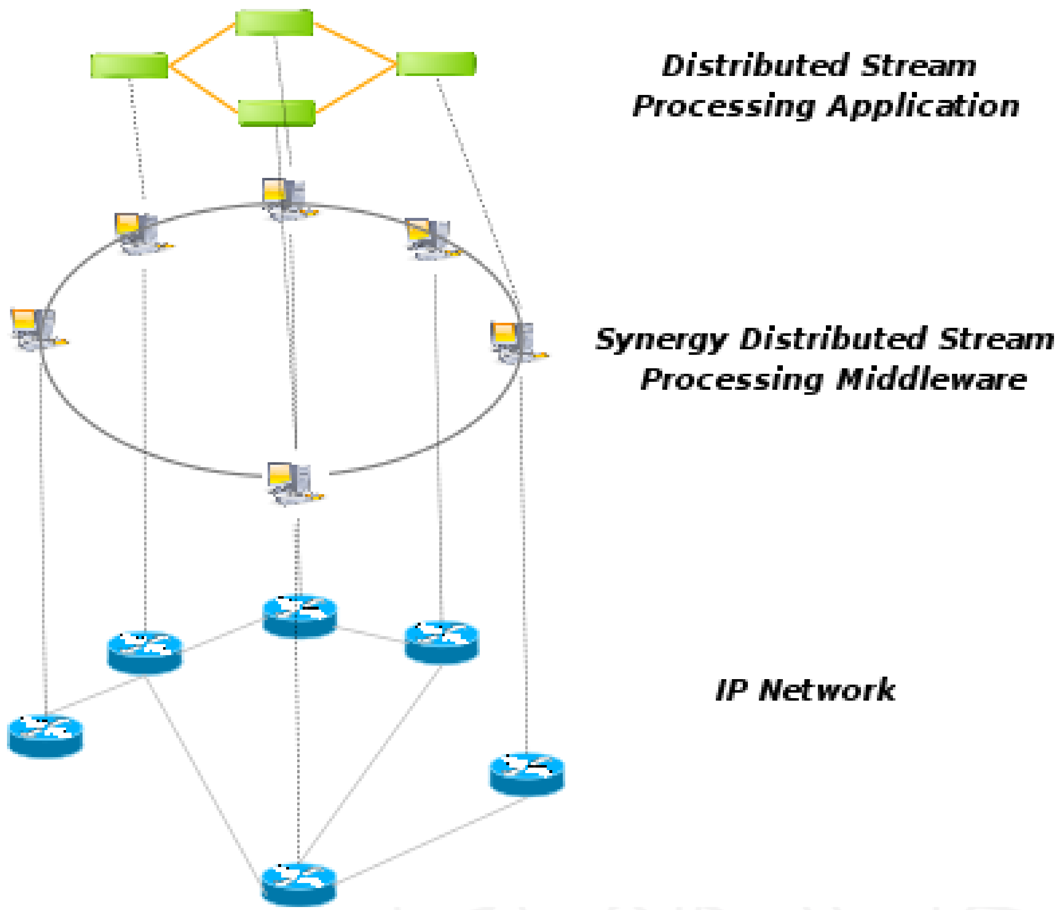
- **Our goal: How to run stream processing applications with QoS requirements, while efficiently managing system resources**
  - Share existing result streams
  - Share existing stream processing components
  - Predict QoS violations
  - Alleviate hot-spots
  - Maximize availability
- **Benefits**
 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

Roadmap

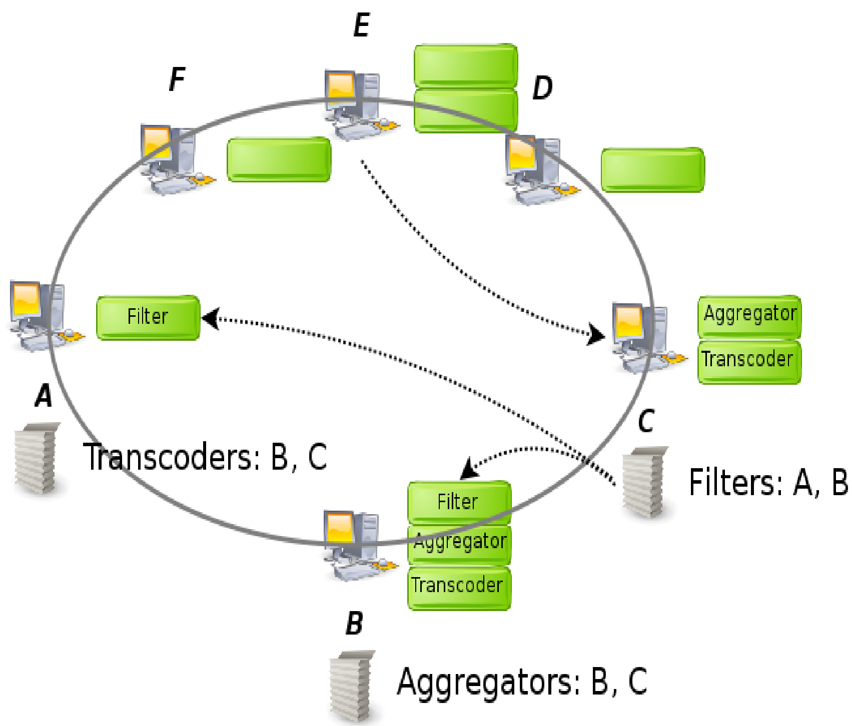
- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Component Composition
    - Composition Protocol
    - Component and Stream Sharing
  - Load Balancing
    - Hot-Spot Prediction
    - Hot-Spot Alleviation
  - High Availability
    - Replica Placement
- Conclusion
- Demo

Synergy Middleware

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

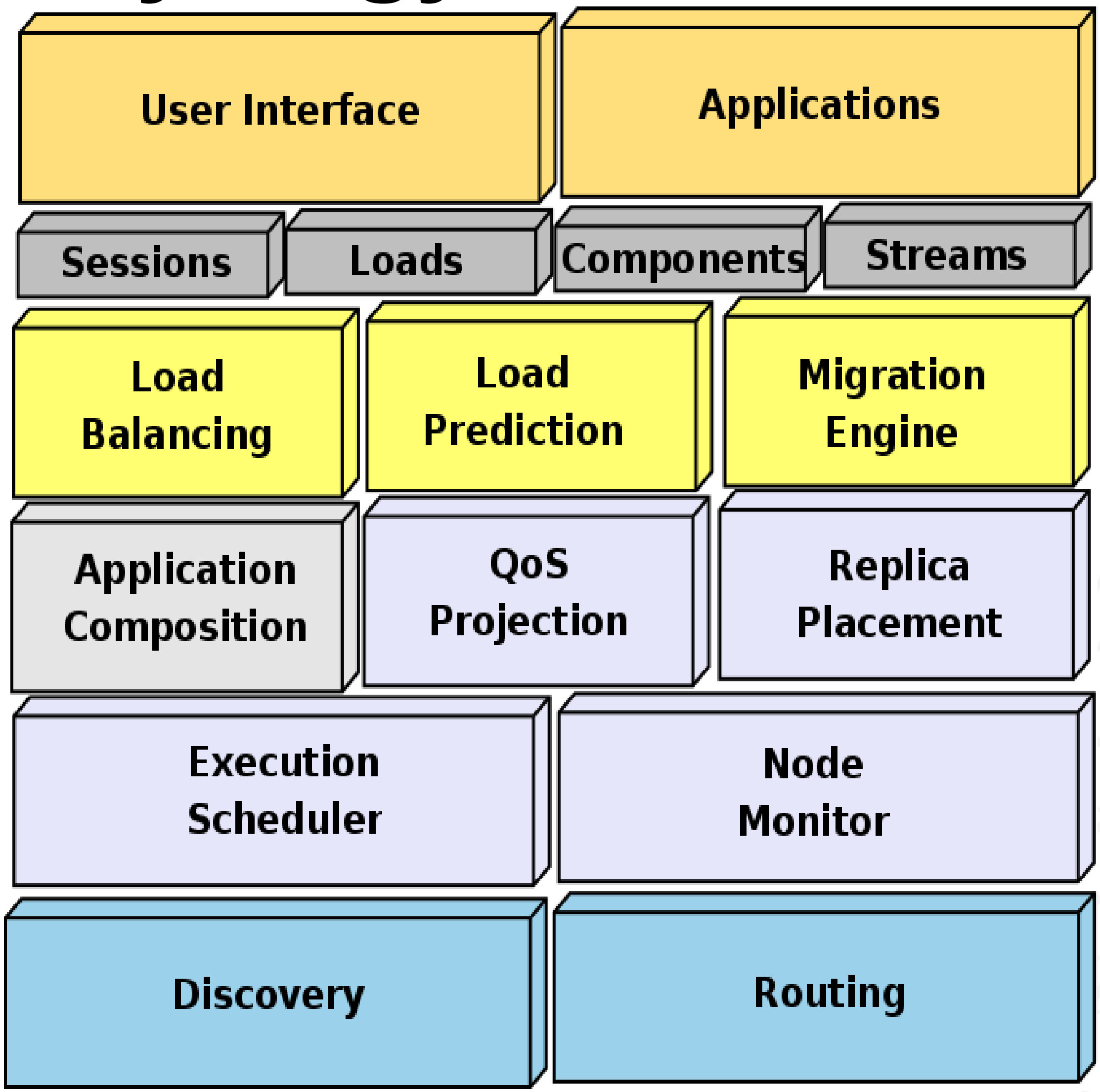


Metadata Layer Over a DHT



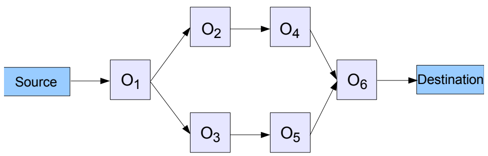
- Decouples stream and component placement from their discovery
- Stream and component names are hashed in a DHT
- DHT maps the hashed names to nodes currently offering the specified stream or component

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

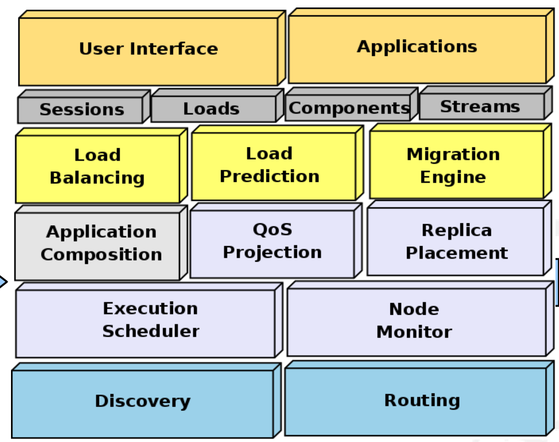
Component Composition



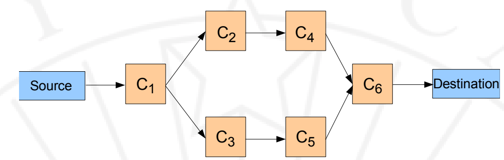
Query Plan

+

QoS Requirements

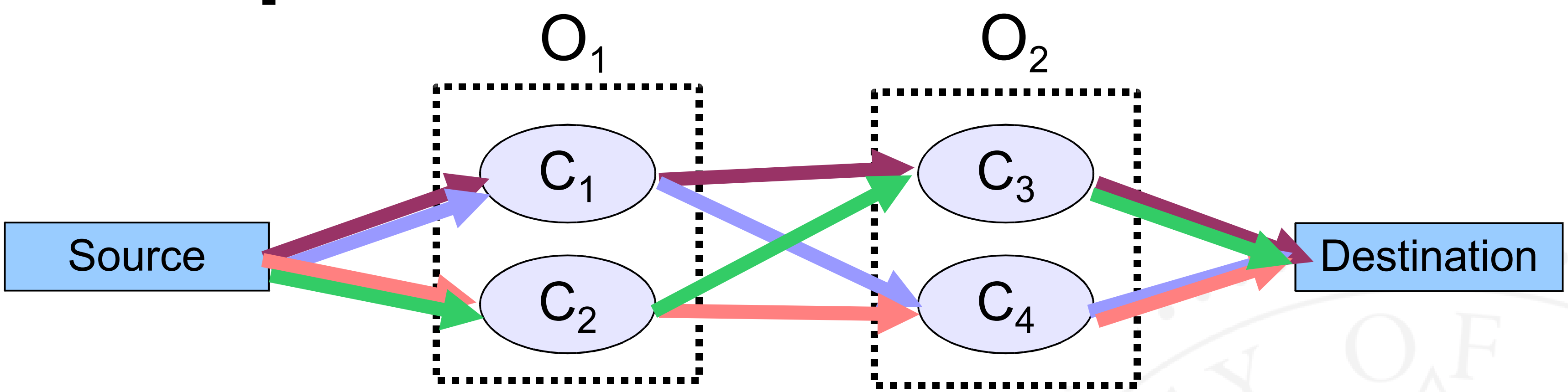


Synergy Middleware



Application Component Graph

Composition Probes



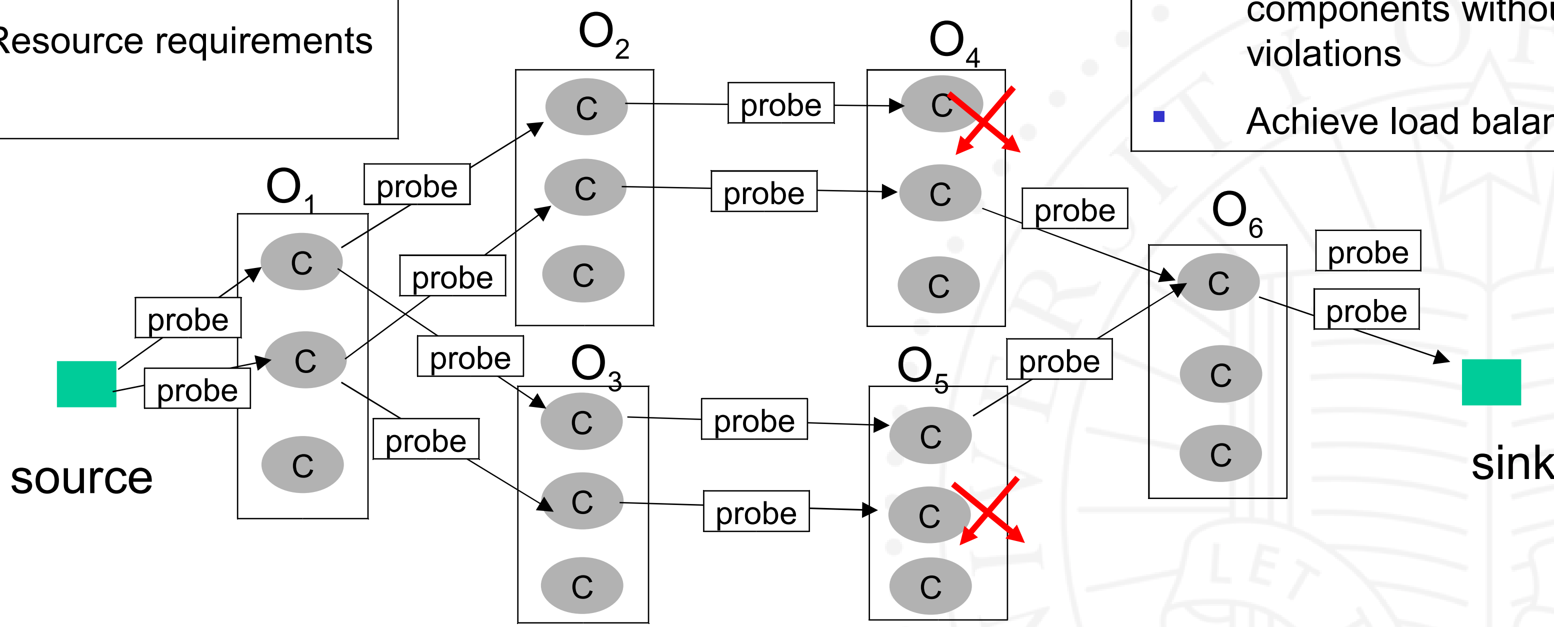
- Carry query plan, resource, and QoS requirements
- Collect information about:
  - Resource availability
  - End-to-end QoS
  - QoS impact on existing applications

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:

$$\phi(\lambda) = \sum_{v_i \in V_\lambda, o_i \in \xi} \frac{p_{o_i}}{rp_{v_i} + p_{o_i}} + \sum_{l_j \in \lambda, s_j \in \xi} \frac{b_{s_j}}{rb_{l_j} + b_{s_j}}$$

Component Sharing

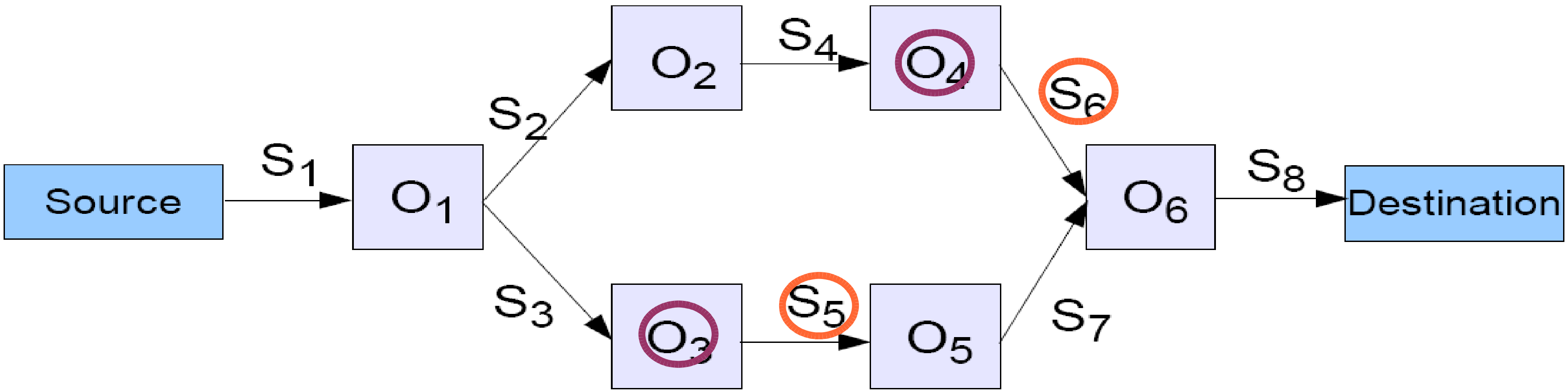
- **QoS Impact Projection Algorithm**
  - All existing and the new application should not exceed requested execution time:

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

  - 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

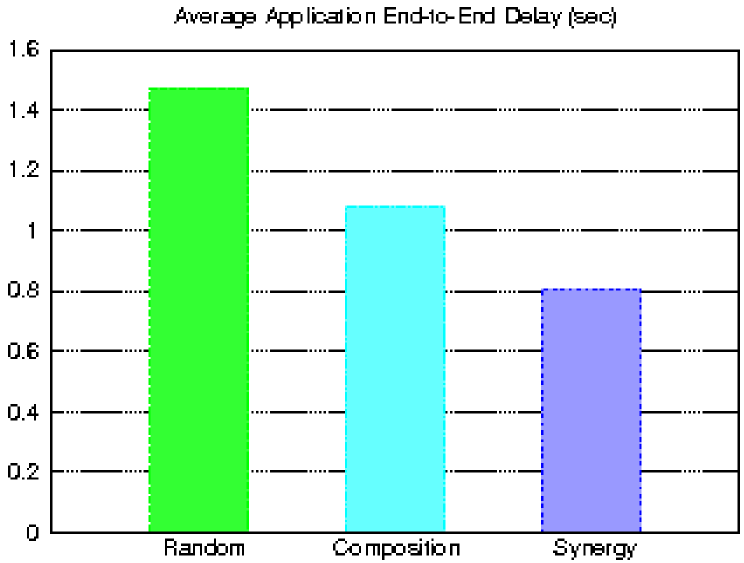


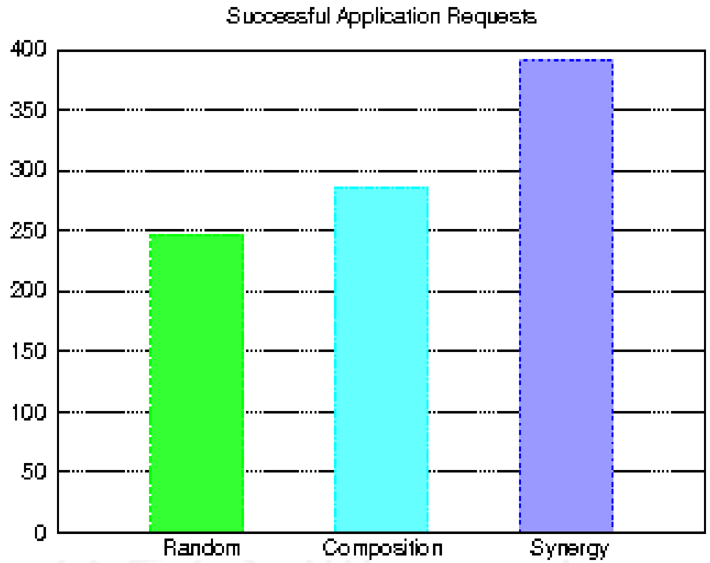
- **Maximum Sharing Discovery Algorithm**
  - Breadth first search on query plan to identify latest possible existing output streams
  - Backtracking hop-by-hop, querying the metadata layer

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

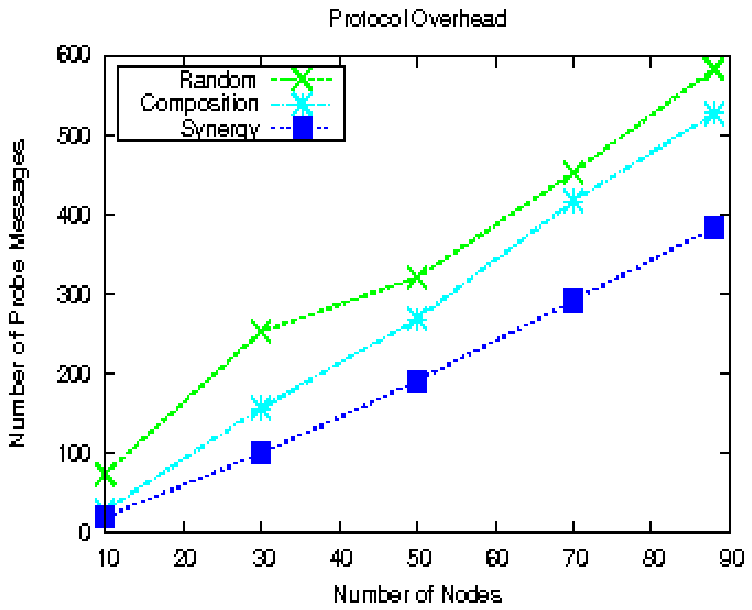
Composition Performance

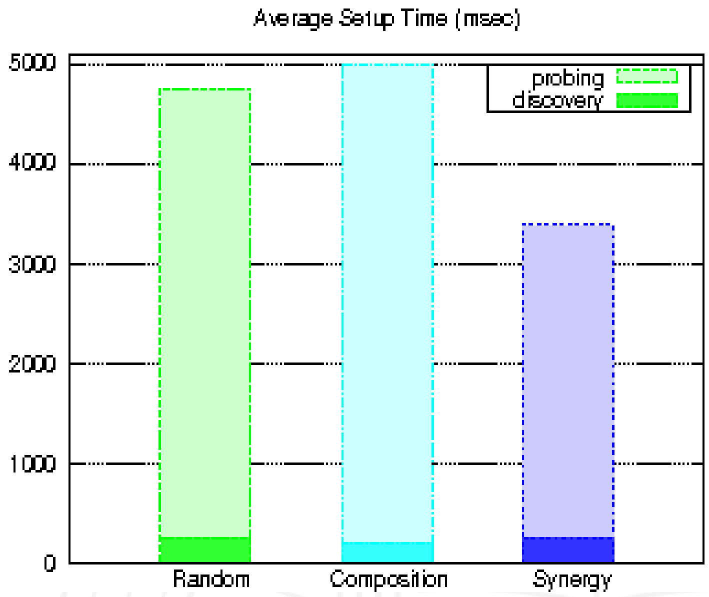




Stream reuse improves end-to-end delay by saving processing time and increases system capacity

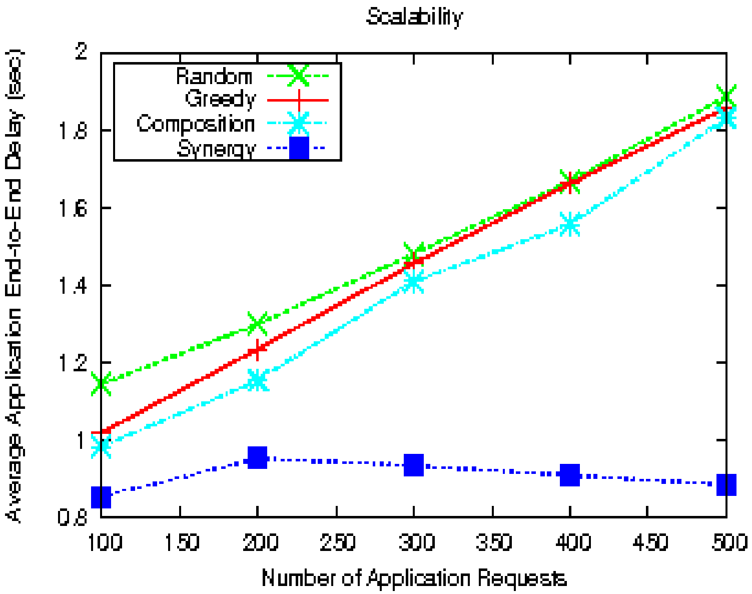
Composition Overhead

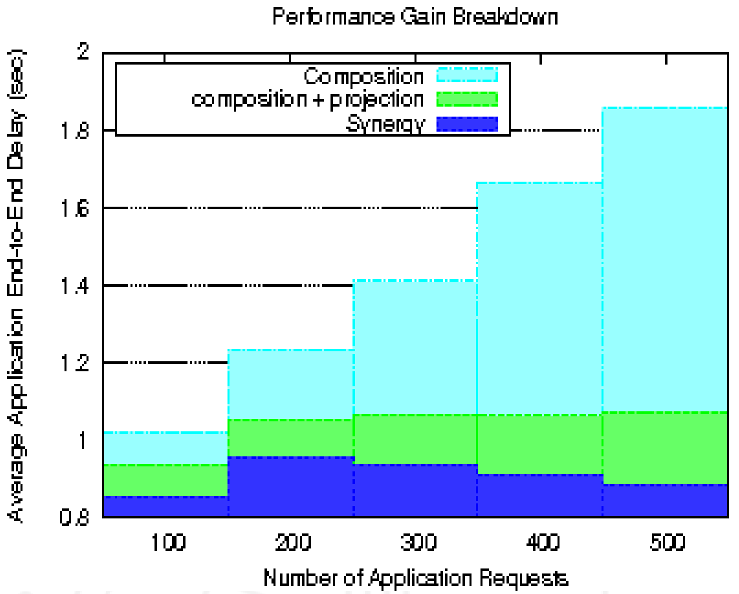




Stream reuse decreases probing overhead and setup time

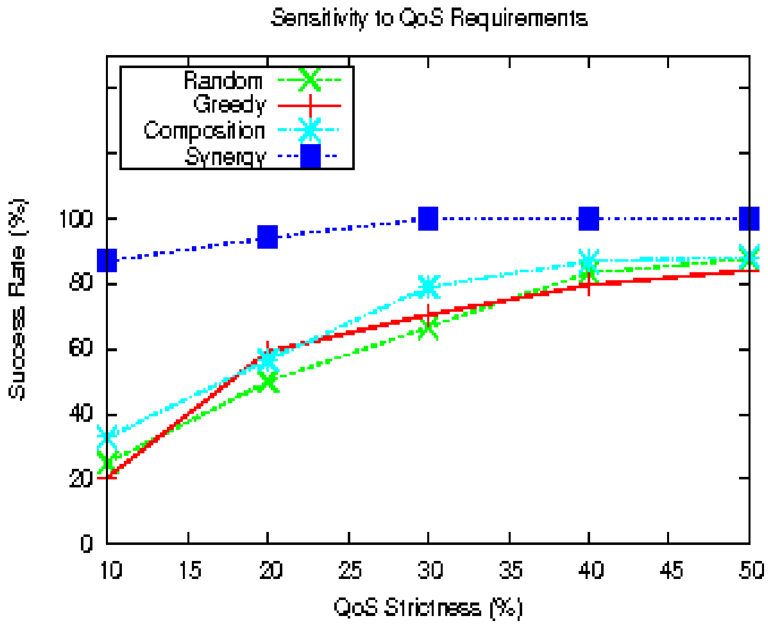
Performance on Simulator

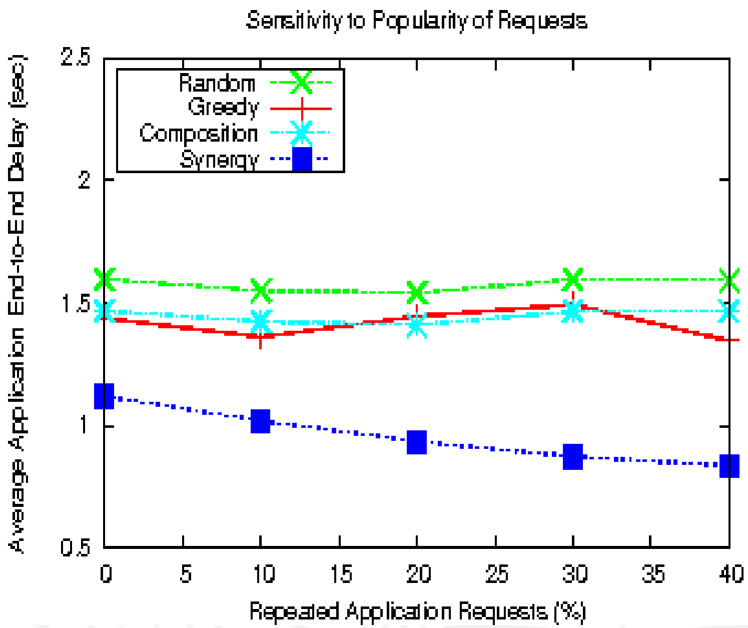




End-to-end delay scales due to stream reuse and QoS impact projection

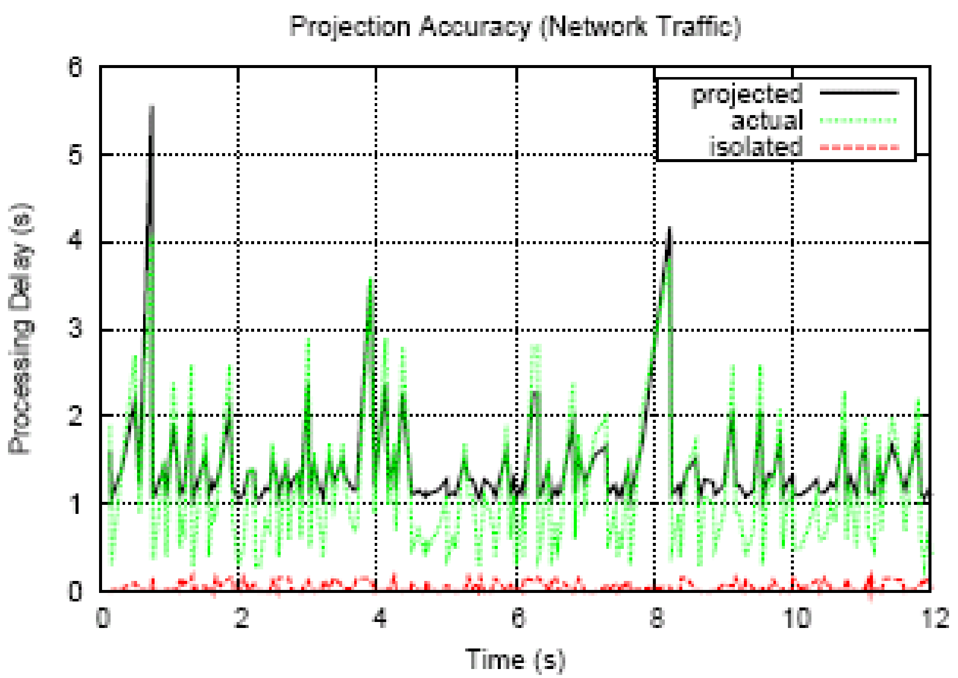
Sensitivity on Simulator

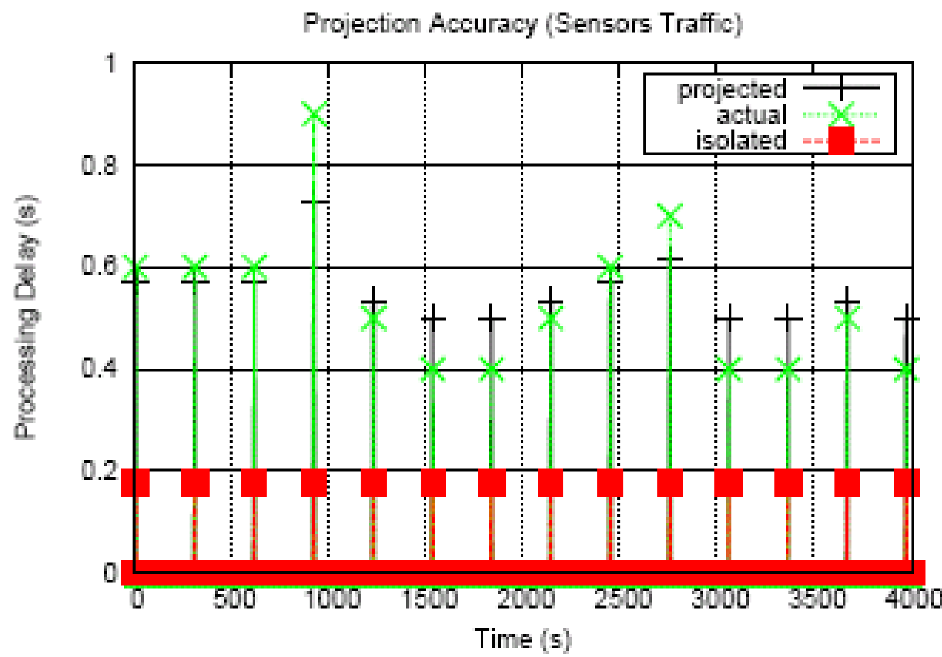




Synergy performs consistently better, regardless of QoS strictness or query popularity

Projection Accuracy





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

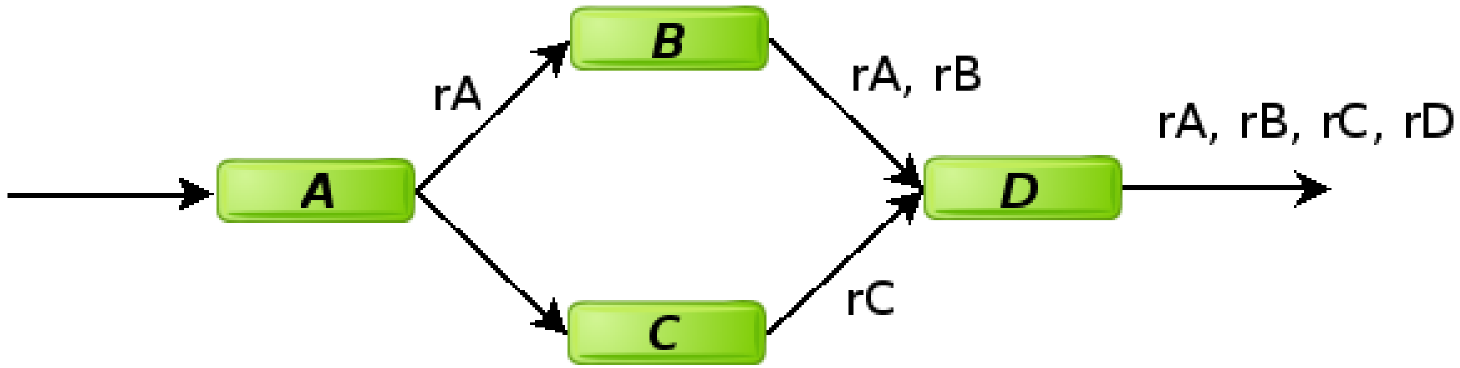
Roadmap

- 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

$$t_{s(i)} = q_t - \Big(\sum_{j \in 1 \ldots i-1} t_{c(j \to j+1)} + \sum_{j \in 1 \ldots i-1} t_{e(j)} + \sum_{i \ldots v-1} \bar{t}_{c(j \to j+1)} + \sum_{j \in i \ldots v} \hat{t}_{e(j)} \Big) > 0$$

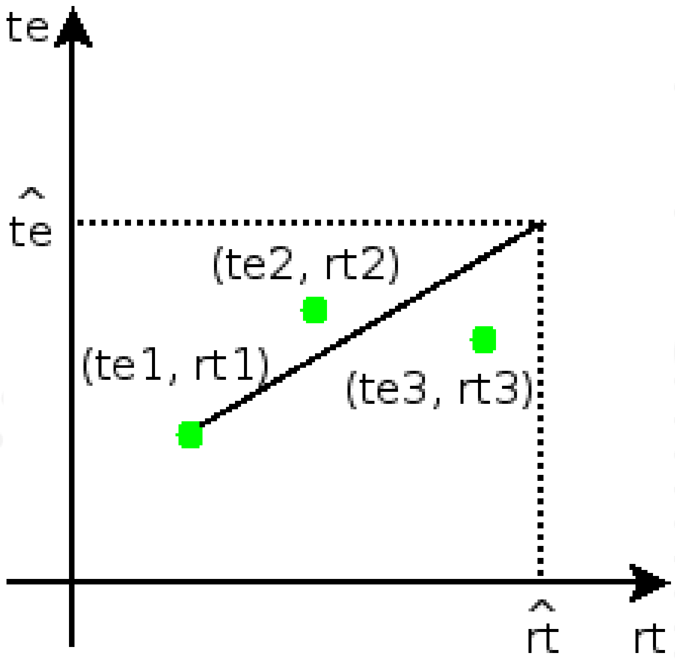
Execution Time Prediction

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

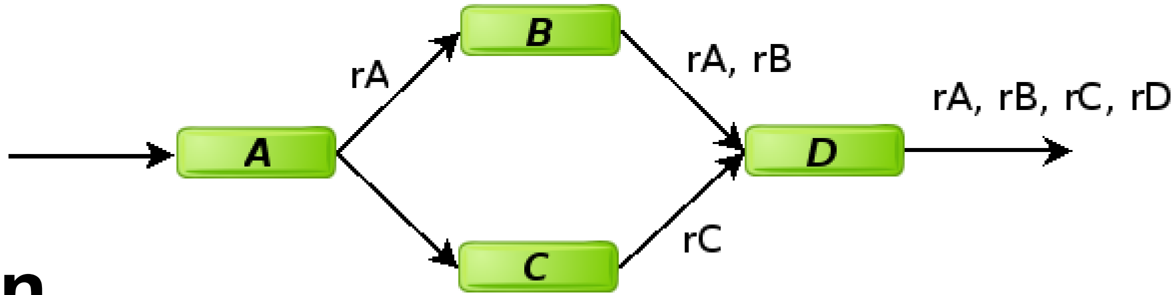
$$\hat{t_e} = a + b \cdot \hat{r_t}$$

$$a = \bar{t_e} - b \cdot \bar{r_t}$$

$$b = \frac{\sum_{j \in 1 \ldots k} (r_{t(j)} - \bar{r_t}) \cdot (t_{e(j)} - \bar{t_e})}{\sum_{j \in 1 \ldots k} (r_{t(j)} - \bar{r_t})^2}$$



Rate Prediction



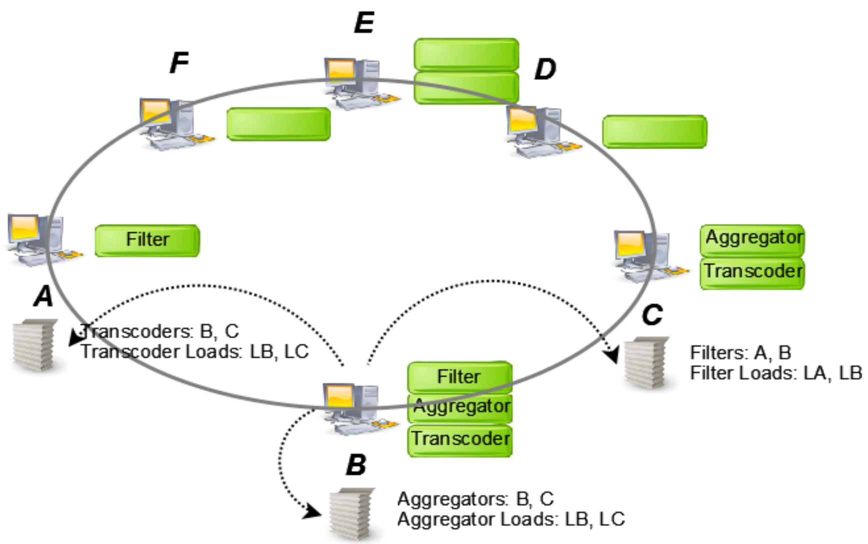
- **Auto-correlation**

$$\hat{r_k} = \frac{\arg_m maxR_{(k)}}{\arg_m maxR_{(k-1)}} \cdot r_{k-1} = \frac{r_{k(m)}}{r_{k-1(m)}} \cdot r_{k-1}$$

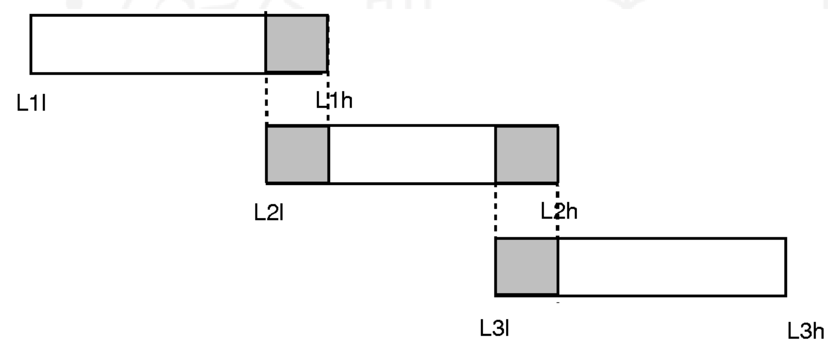
- **Cross-correlation (Pearson Product Moment)**

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

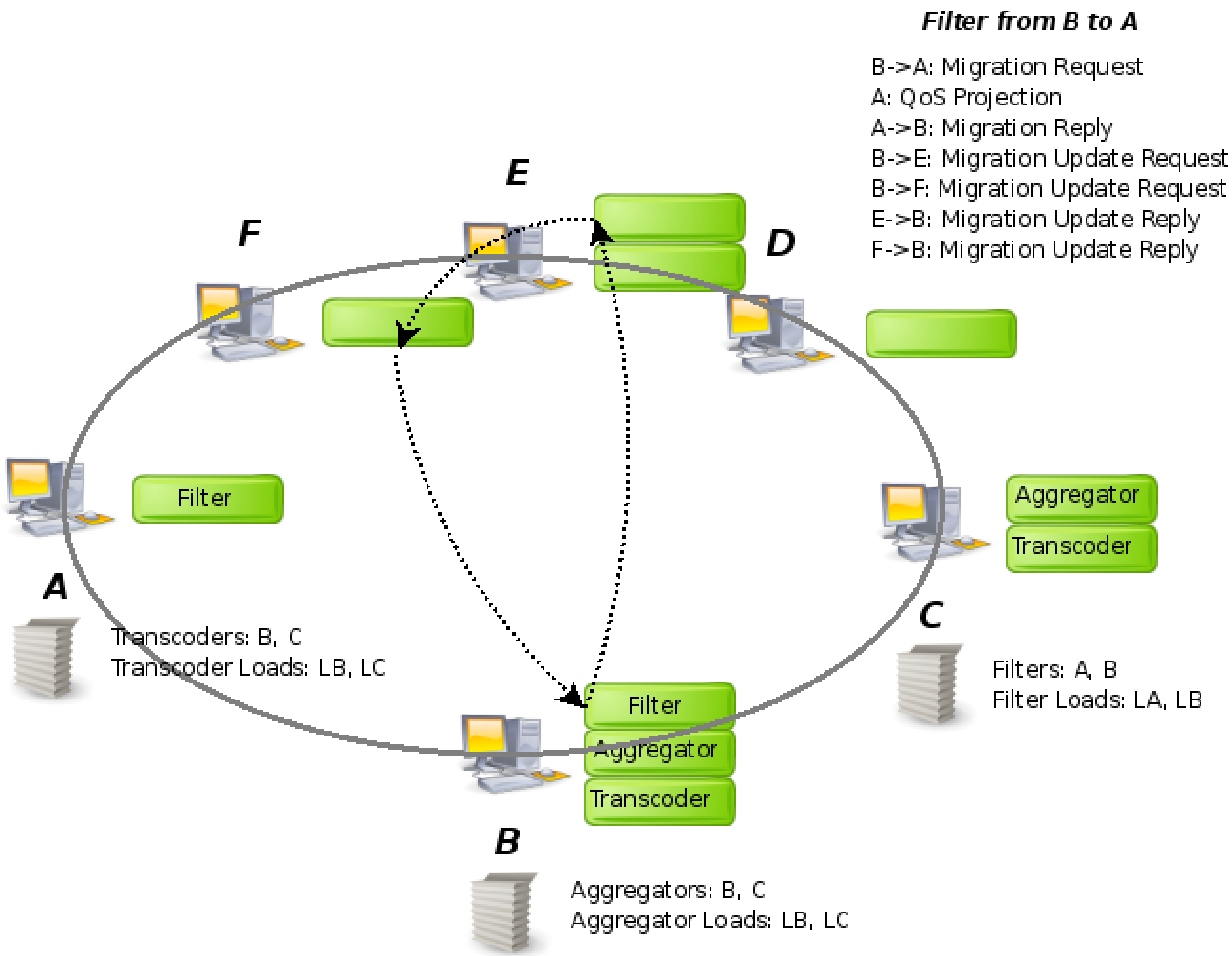
Decentralized Load Monitoring



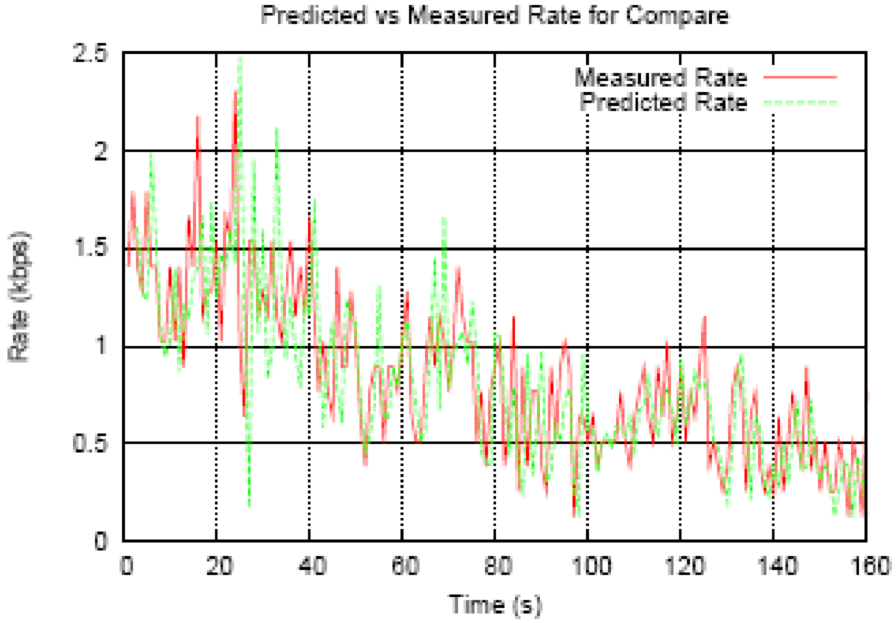
- DHT maps component names to the loads of peers hosting them
- Peers detect overloads and imbalances between all hosts of a component
- Load updates pushed when intervals change
- Overlapping intervals absorb frequent changes

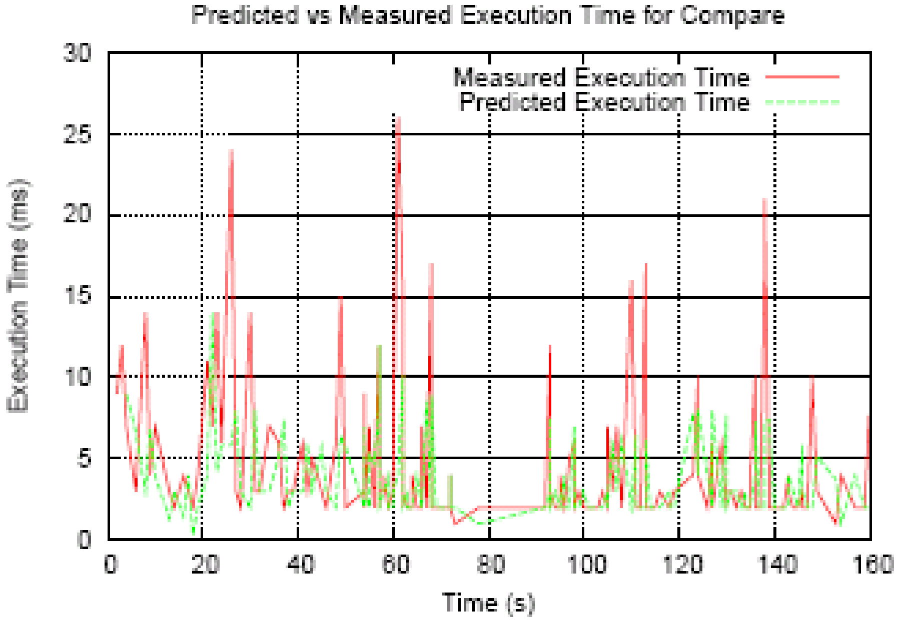


Alleviating Hot-Spots via Migration



Hot-Spot Prediction and Alleviation

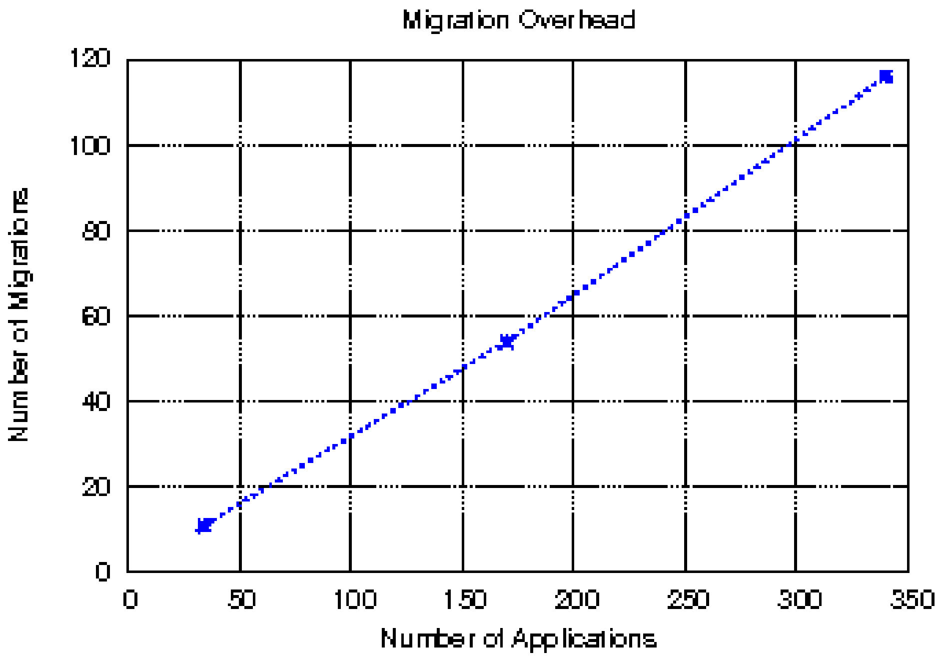


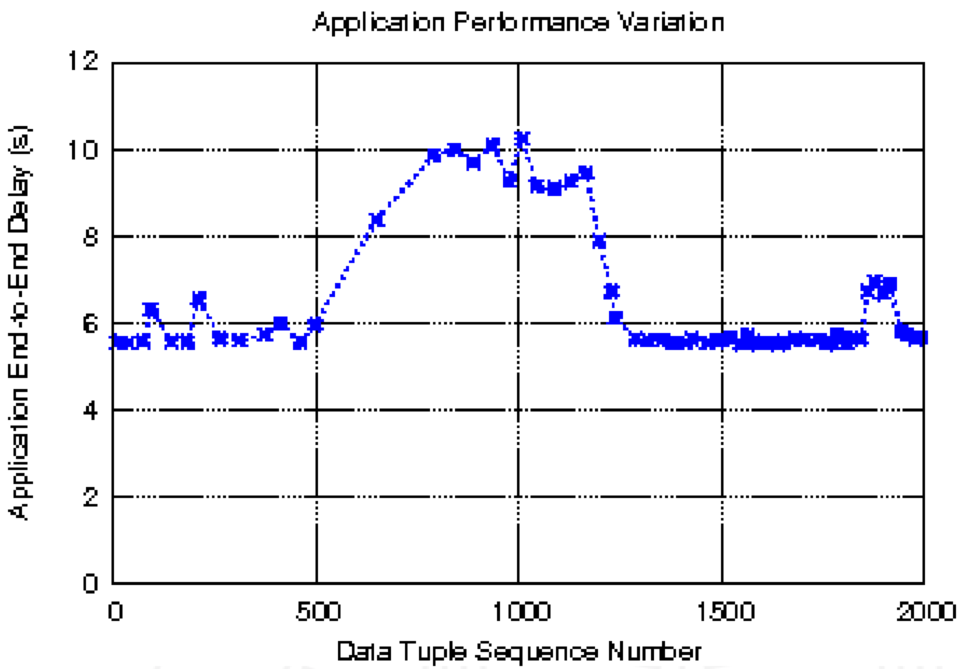


Average prediction error 3.7016%

Average prediction overhead 0.5984ms

Hot-Spot Prediction and Alleviation

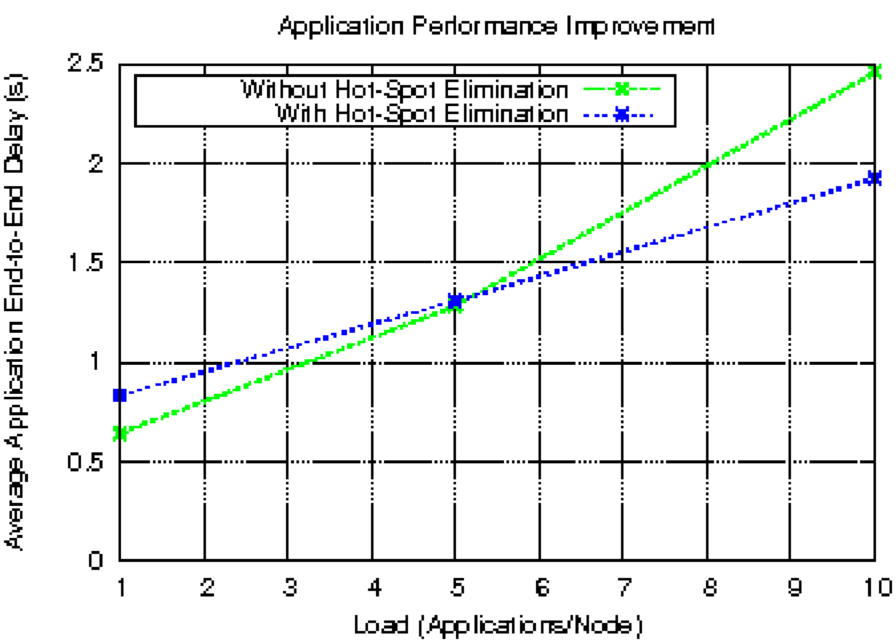


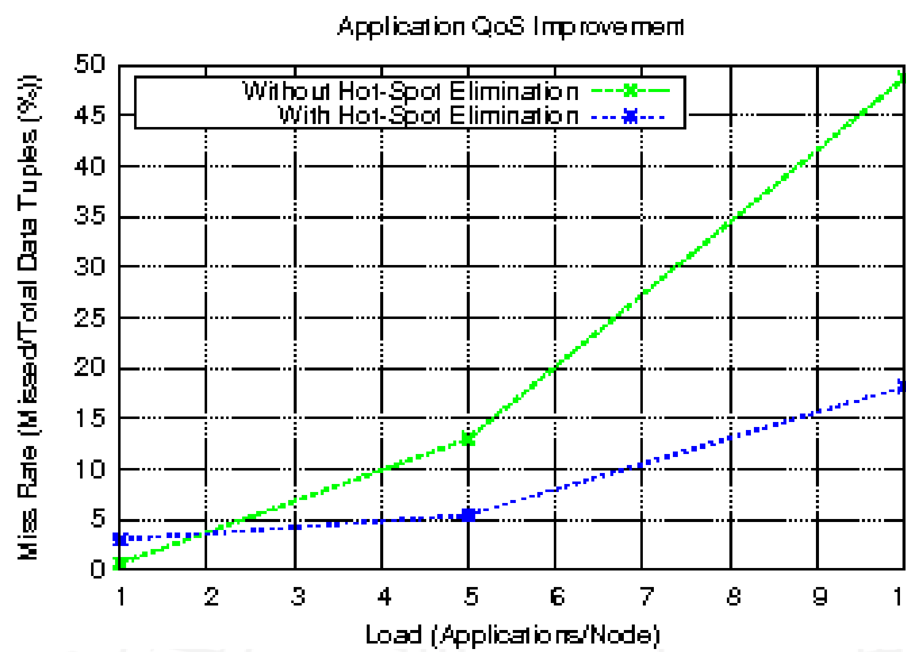


Average one migration every three applications

Average migration time 1144ms

QoS Improvement



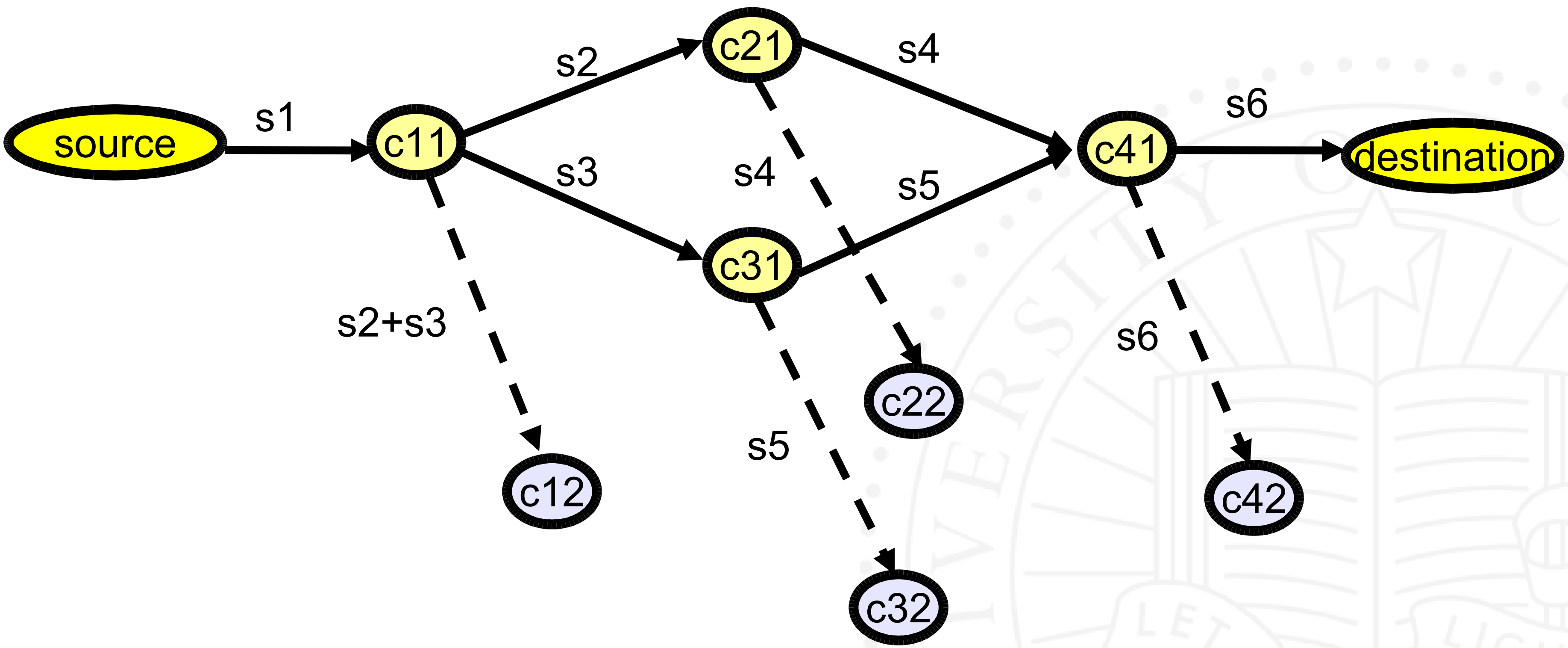


As load increases the benefits of hot-spot elimination become evident

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

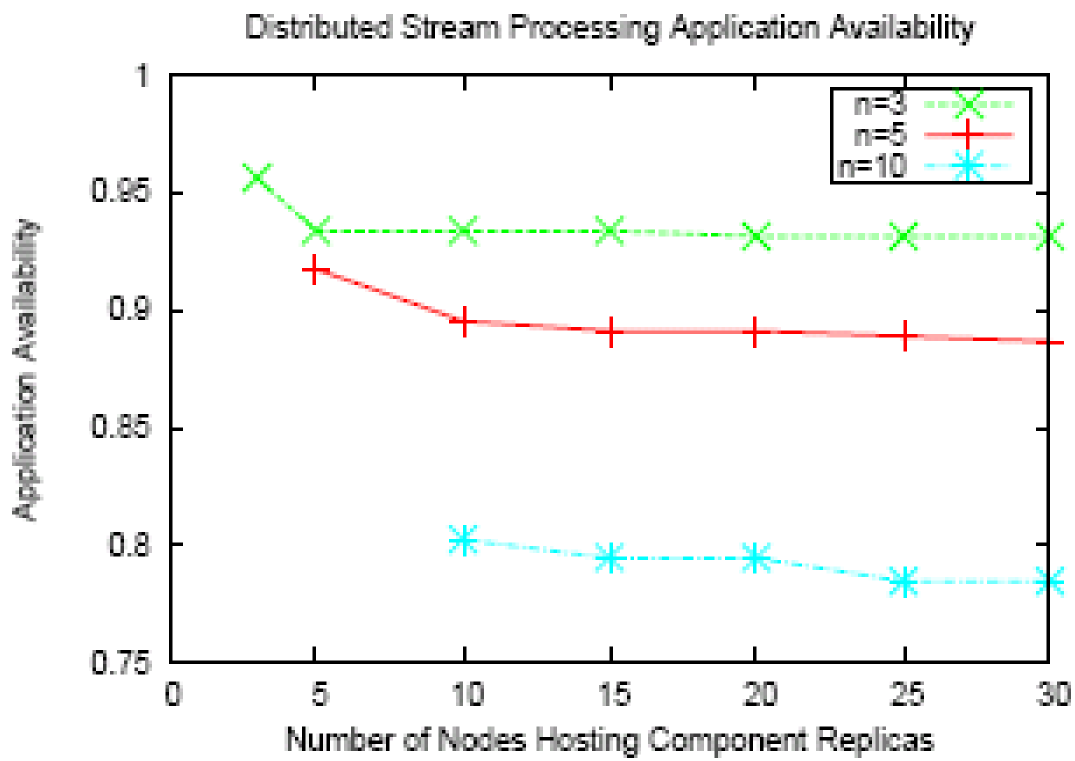
Component Replication



Component Replica Placement

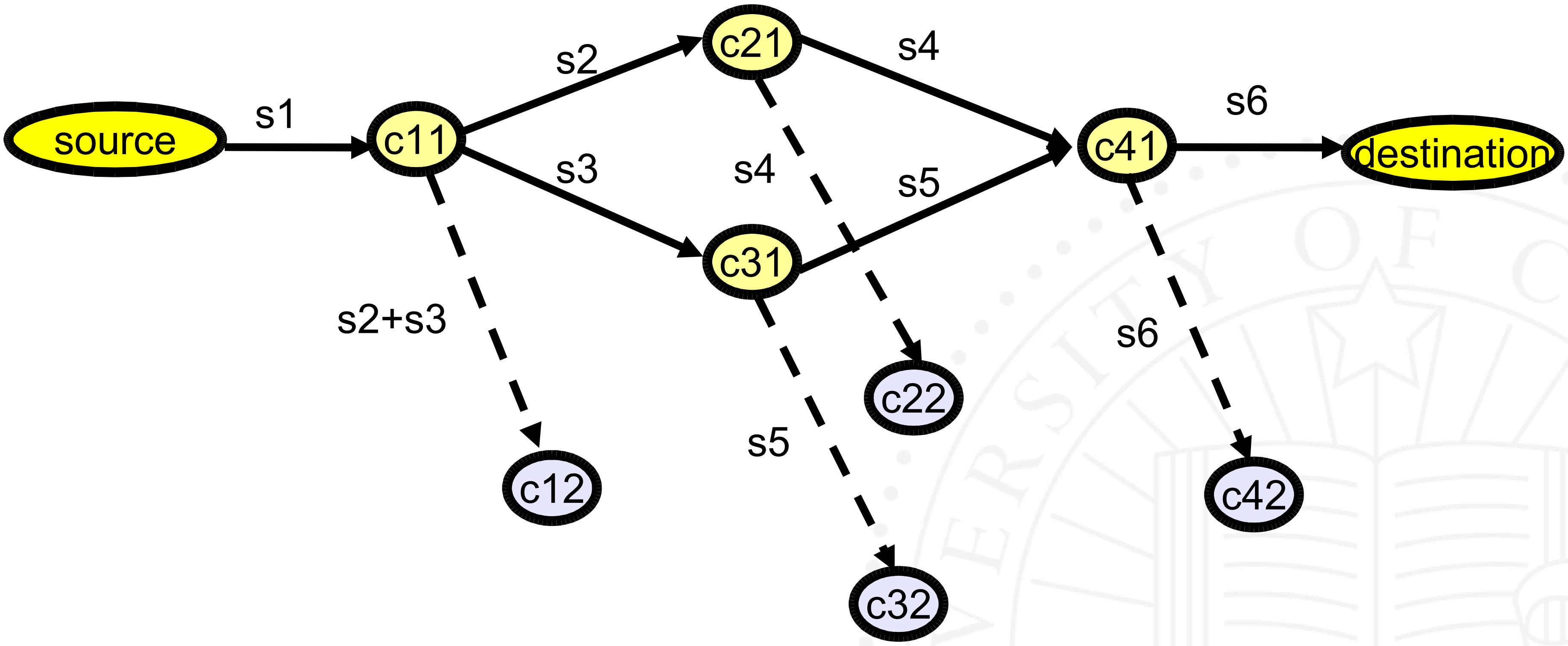
- Maximize availability of composite applications
  - Optimal: Place complete graph on each node
- Respect node resource availability
  - Processing capacity
  - Network bandwidth
- Maximize application performance
  - Inter-operator communication cost (between primaries)
  - Intra-operator communication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

Placement for High Availability



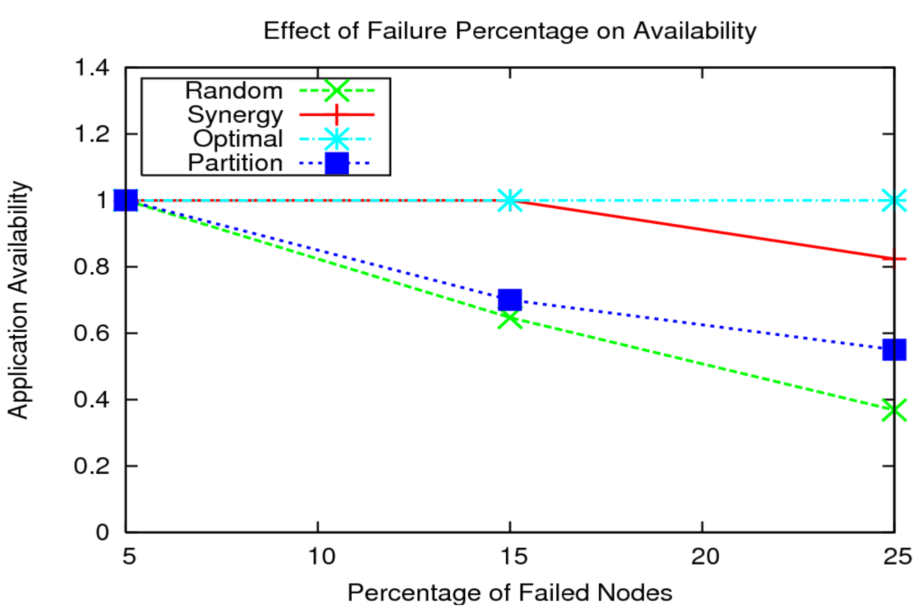
Availability decreases with larger graphs and increases with higher concentration

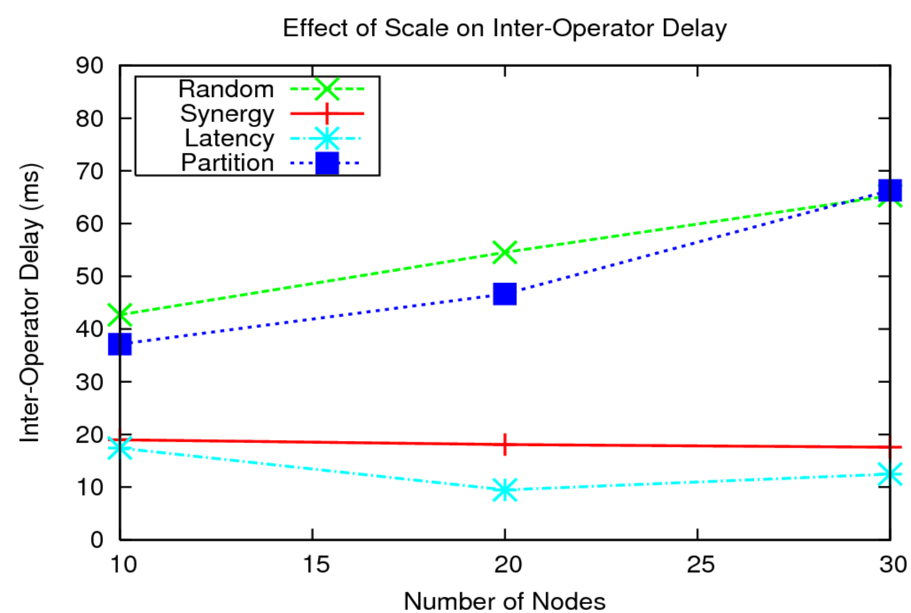
Distributed Placement Protocol



Closest used candidates

Replica Placement





Increase availability and performance

5539ms to gather latencies for 30 nodes

Related Work

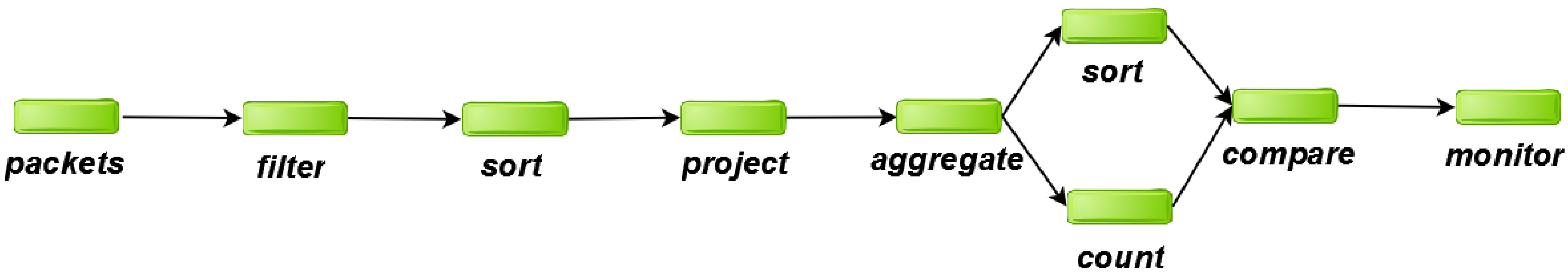
- System S: IBM stream processing middleware
- SBON, SAND, IFLOW: Component placement
- Borealis, Flux, PeerCQ: Load balancing
- Borealis, TelegraphCQ: Load shedding
- Borealis, Flux: Fault tolerance
- SpiderNet, sFlow: Component composition

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

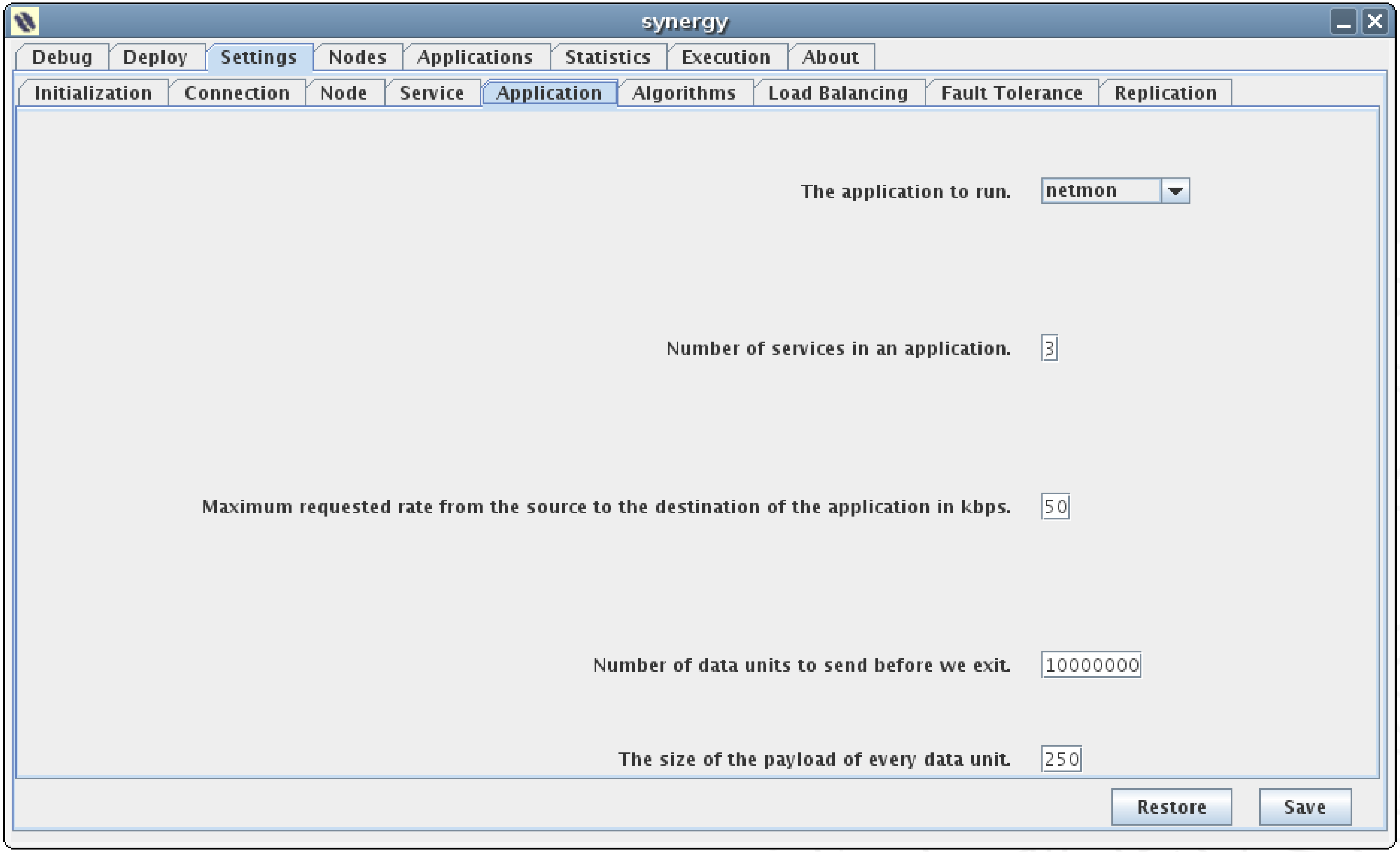
- Monitor source-destination pairs in top 5% of total traffic over last 20 minutes [Stream Query Repository]



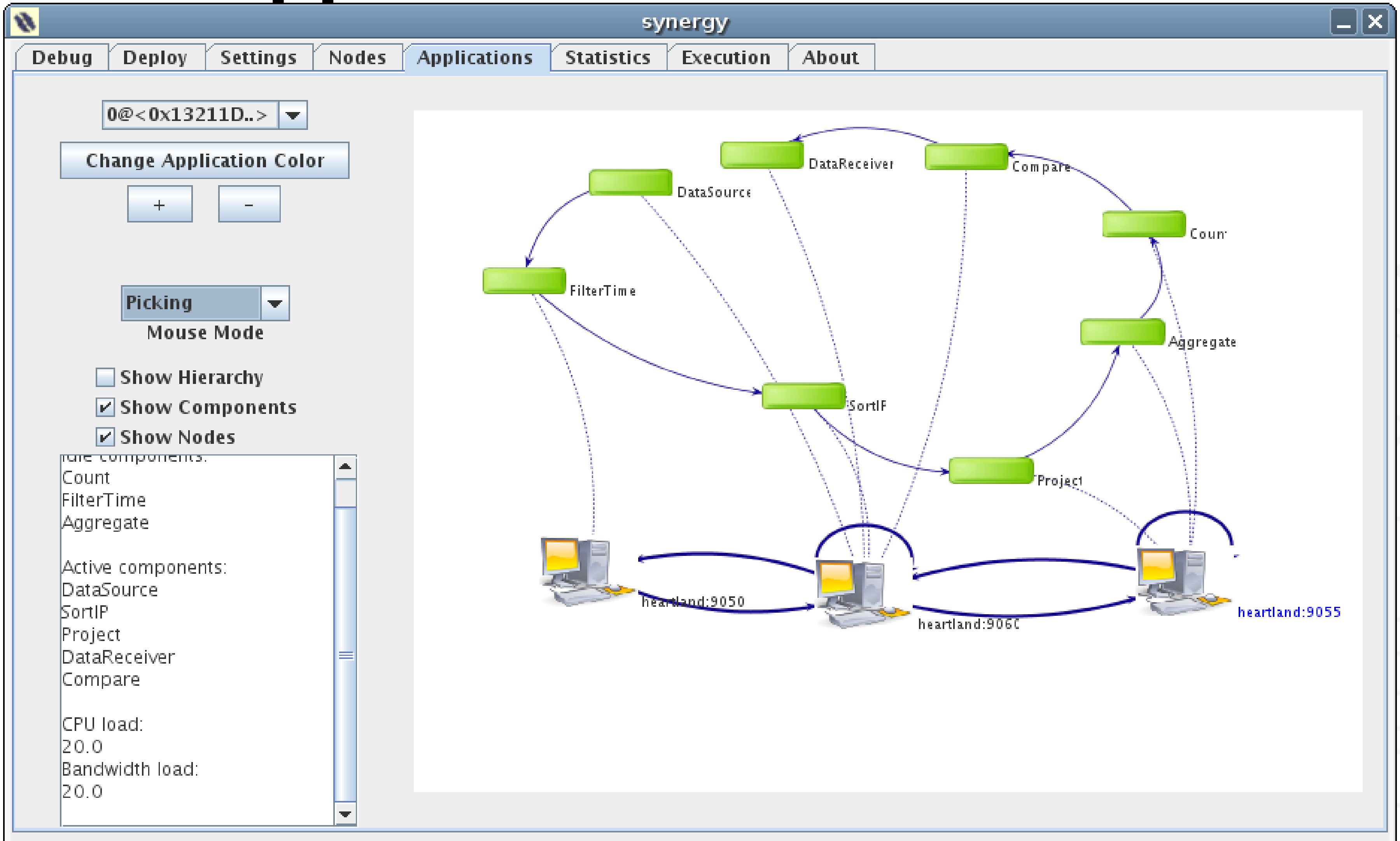
- TCP traffic trace, LBL, 2 hours, 1.8 million packets [Internet Traffic Archive]

timestamp	sourceIP	destinationIP	sourcePort	destinationPort	size

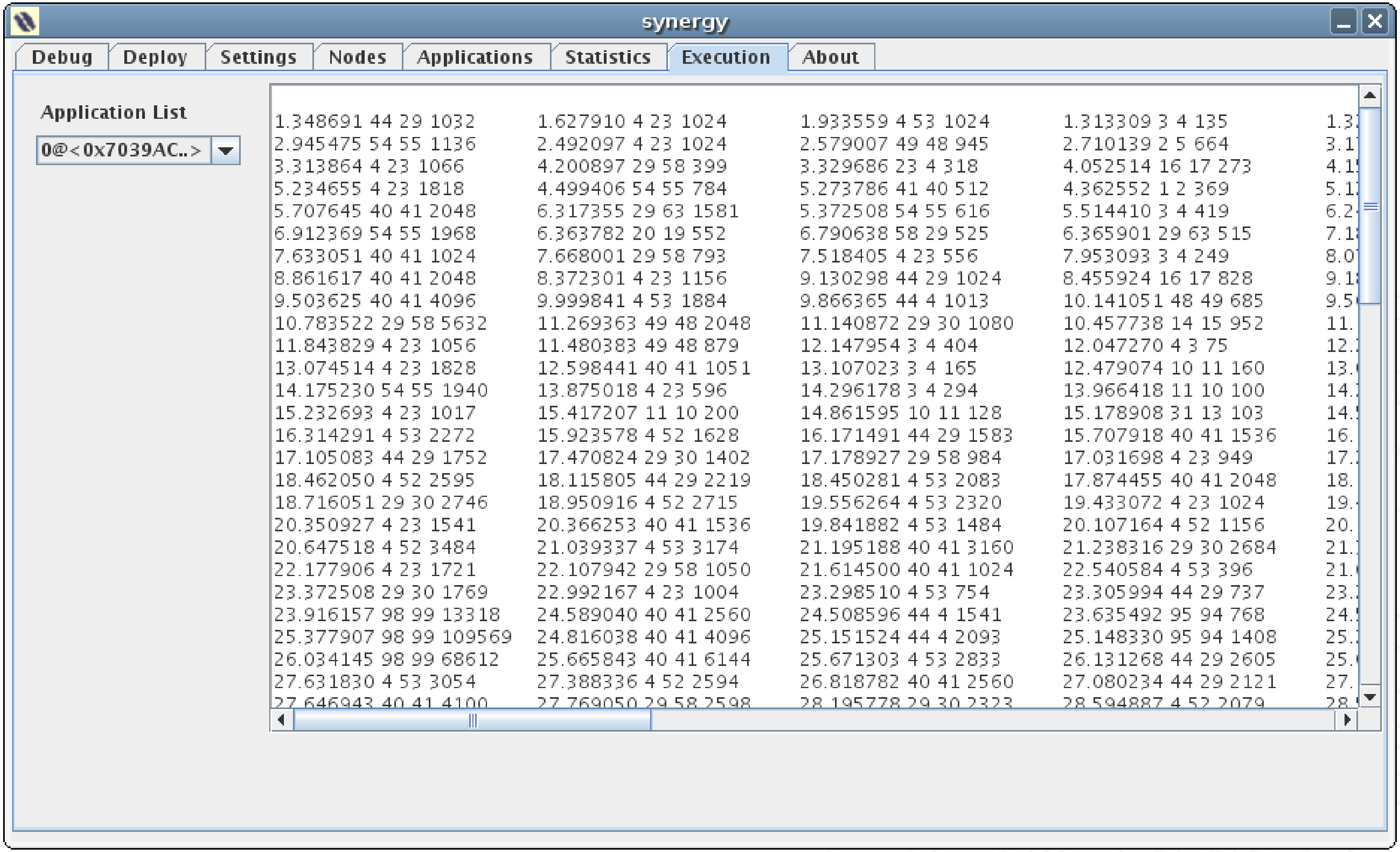
GUI Settings



GUI Application



GUI Execution



Acknowledgements

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

synergy

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

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/

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE