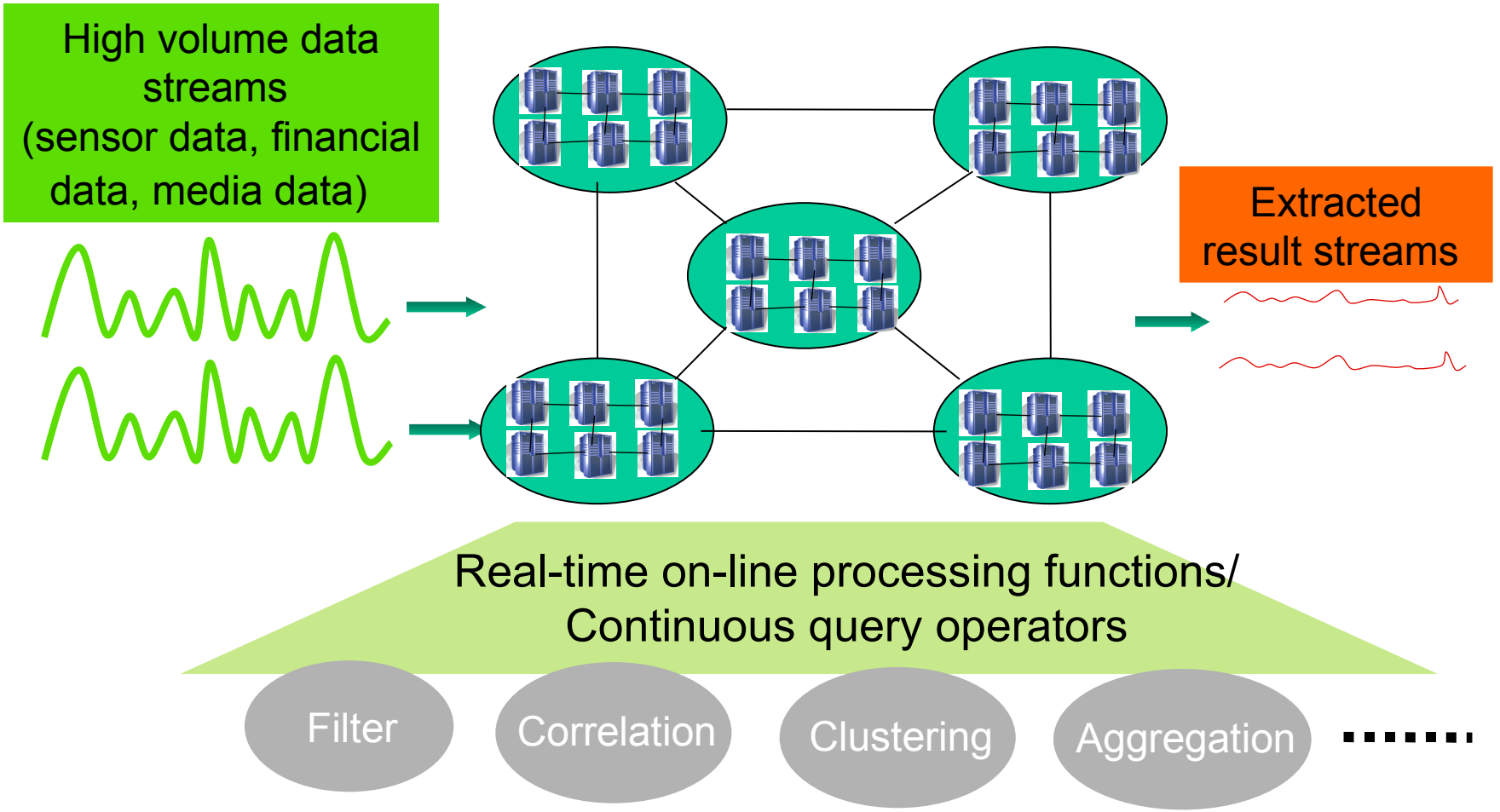
Synergy: Sharing-Aware Component Composition for Distributed Stream Processing Systems

Thomas Repantis[1], *Xiaohui Gu*[2], *Vana Kalogeraki*[1]

trep@cs.ucr.edu, xiaohui@us.ibm.com, vana@cs.ucr.edu
[1] University of California, Riverside
[2] IBM T.J. Watson Research Center, New York

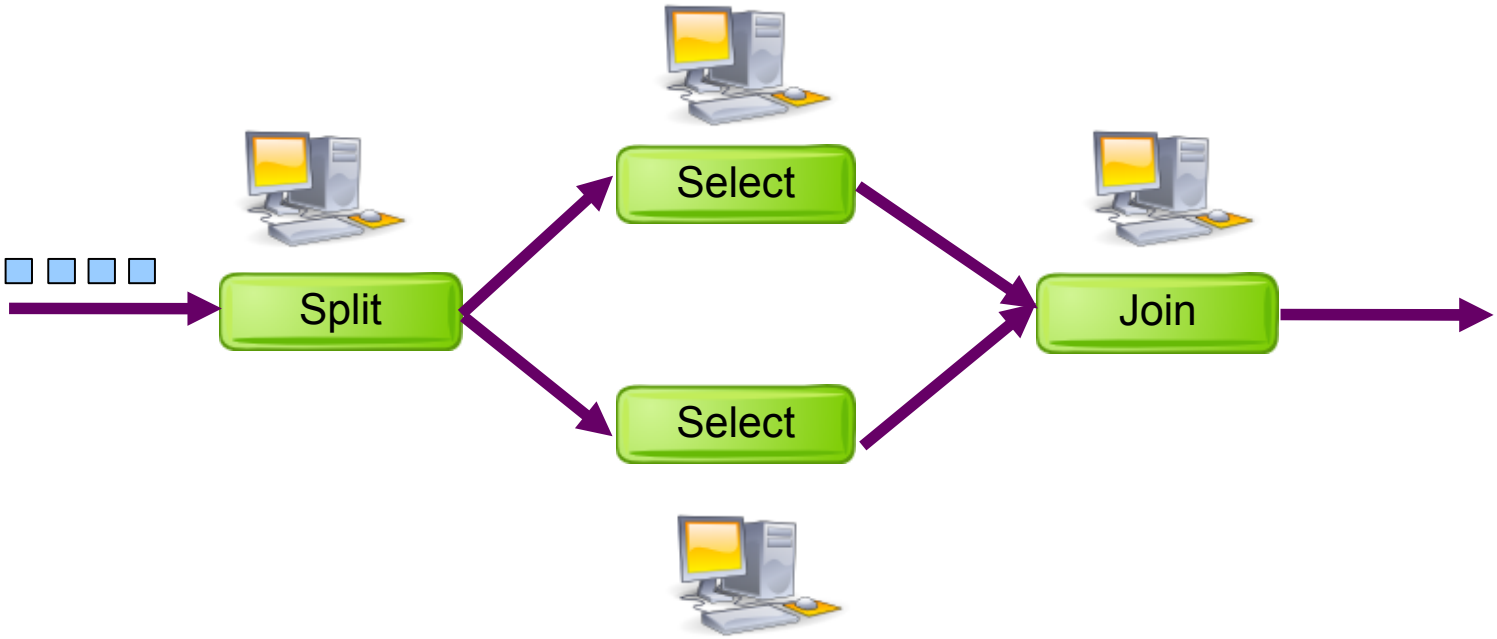
On-line Data Stream Processing



Stream Processing Applications

- Network traffic monitoring for intrusion detection
- Click stream analysis for purchase recommendations
- Financial trading surveillance for fraud detection
- Sensor data analysis for environment monitoring
- Many more…

Stream Processing Environment



- Data arrive in large volumes and high rates
- Streams are processed by components distributed across hosts
- Stream processing applications have QoS requirements, e.g., e2e delay

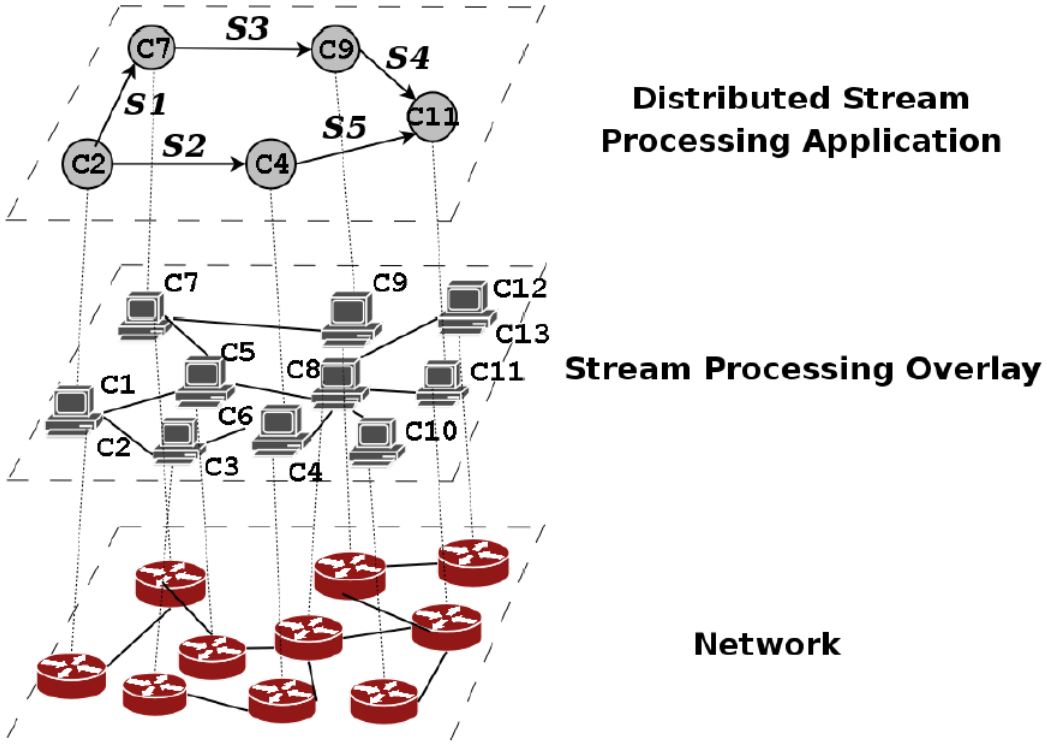
Sharing-Aware Component Composition

- **Our goal: How to compose stream processing applications with QoS requirements**
  - Share existing result streams
  - Share existing stream processing components
- **Benefits**
  - Enhanced QoS provision
  - Reduced resource load
- **Challenges**
  - Concurrent component sharing
  - Highly dynamic environments
  - On-demand stream application requests
  - Scale that dictates decentralization

Roadmap

- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Composition Protocol
  - Component Sharing
  - Stream Sharing
- Experimental Evaluation
- Related Work
- Conclusion

Synergy Middleware

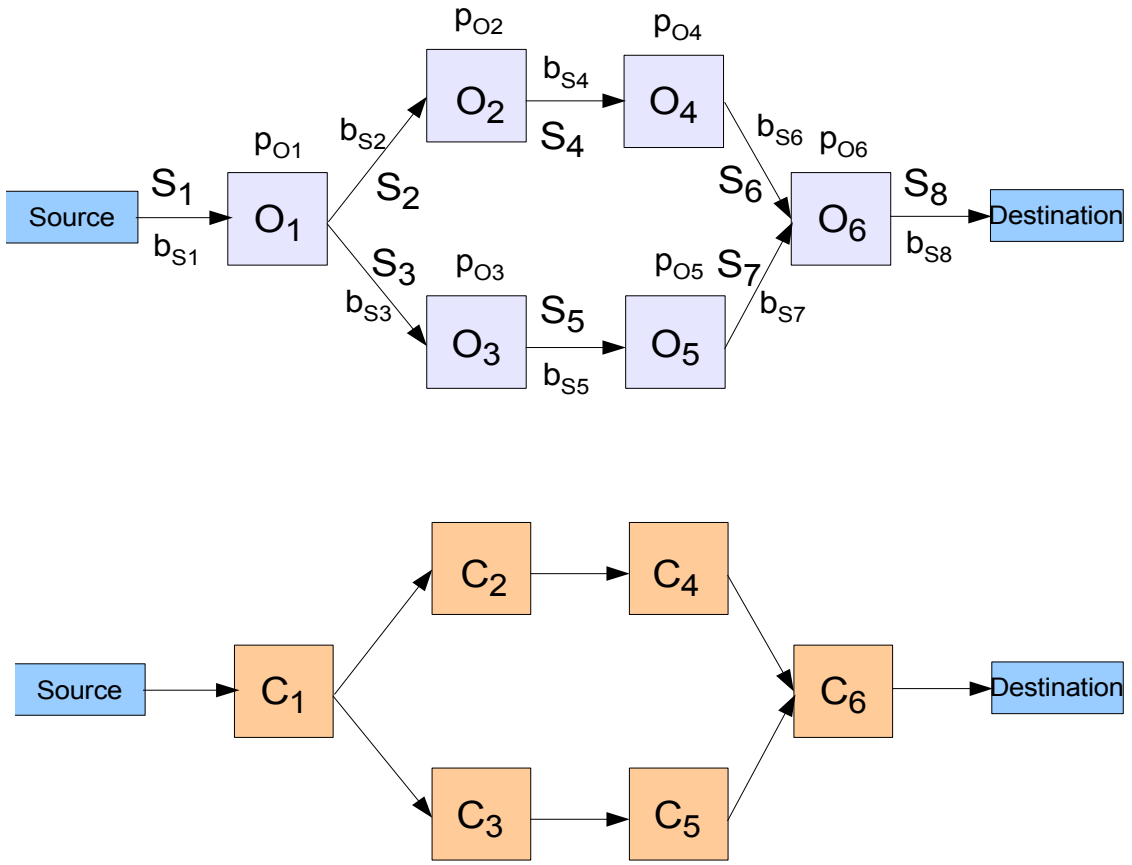


- A middleware managing the mappings:
  - From application layer to stream processing overlay layer
  - From stream processing overlay layer to physical resource layer

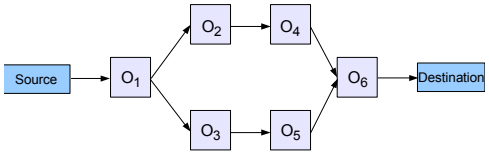
Application Model

Query plan

Instantiated application component graph



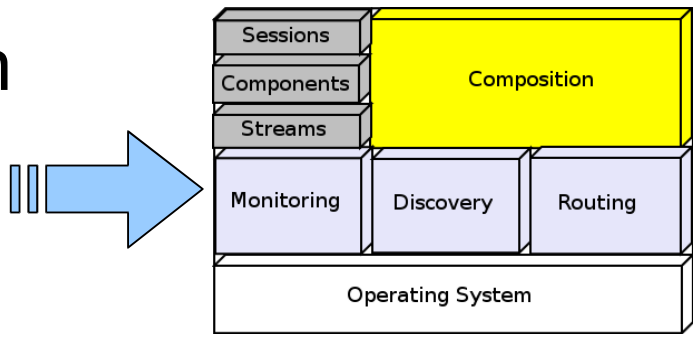
Component Composition



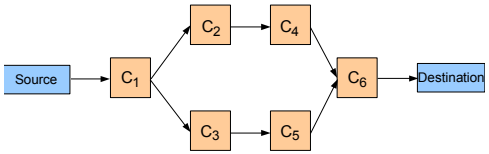
Query Plan

+

QoS Requirements

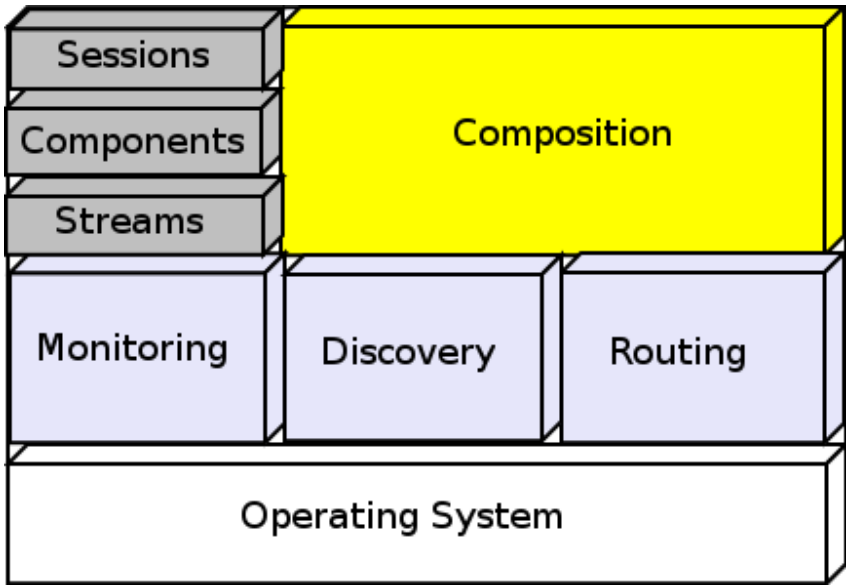


Synergy Middleware



Application Component Graph

Synergy Node Architecture

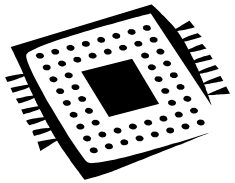


- Composition module runs the component composition algorithms
- Monitoring module observes processor and bandwidth utilization
- Discovery module locates streams and components
- Routing module transfers streaming data

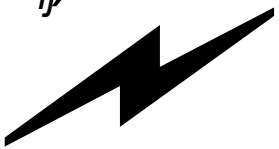
Resource Monitoring

Every node monitors its resource utilization

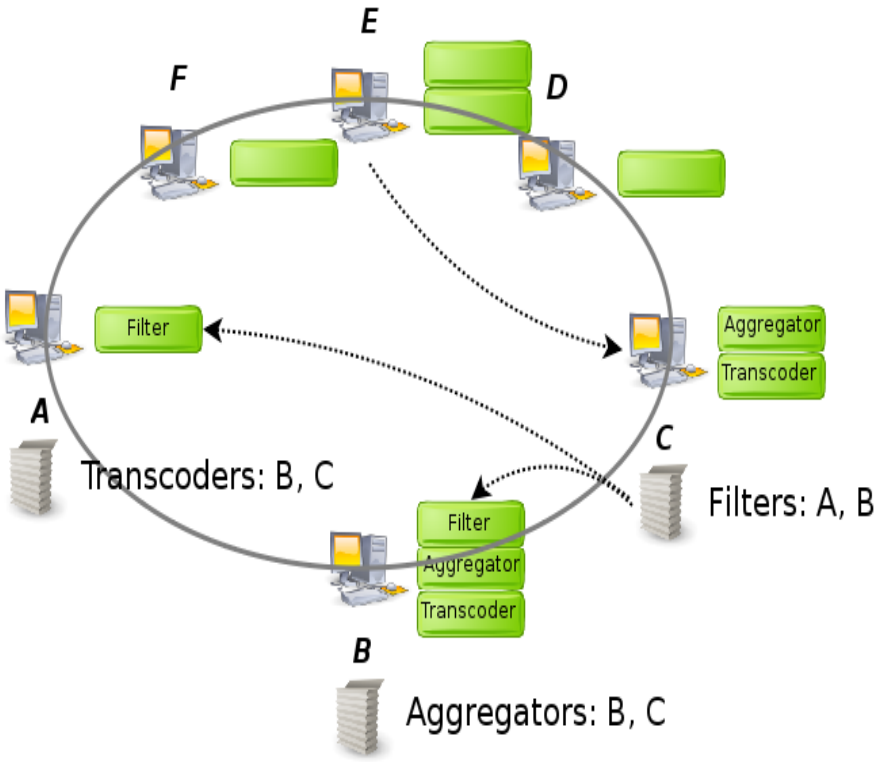
- Residual processing capacity of the node, inferred from the CPU idle time (rp_{vi})
  - *Read from /proc interface*



- Residual bandwidth of each virtual link, calculated by a bandwidth measuring tool (rb_{lj})
  - E.g. Iperf, SProbe, BRoute

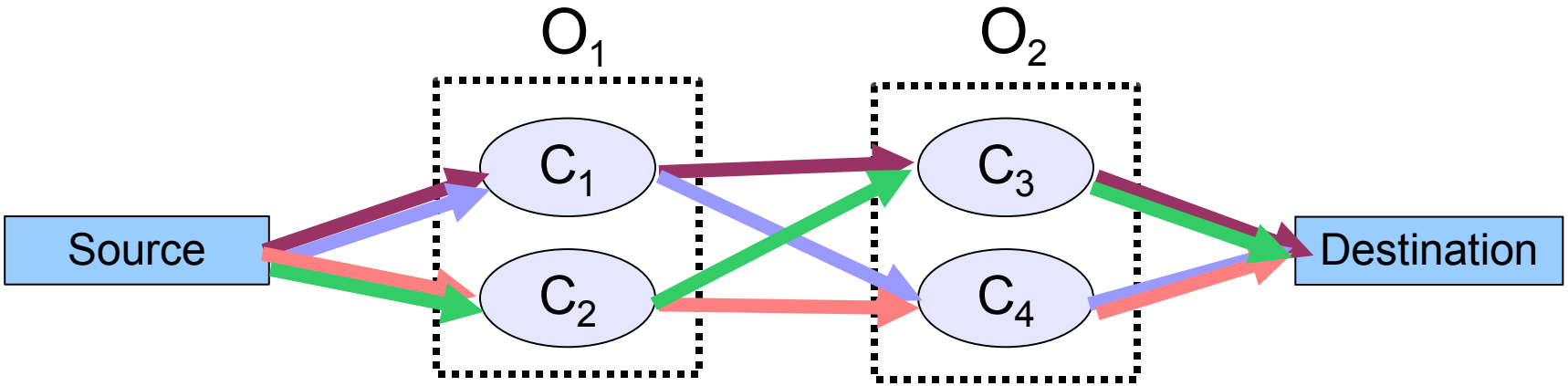


Metadata Layer over a DHT



- Stream and component names are hashed in a DHT
- DHT maps the hashed name to nodes currently offering the specified stream or component

Composition Probing



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

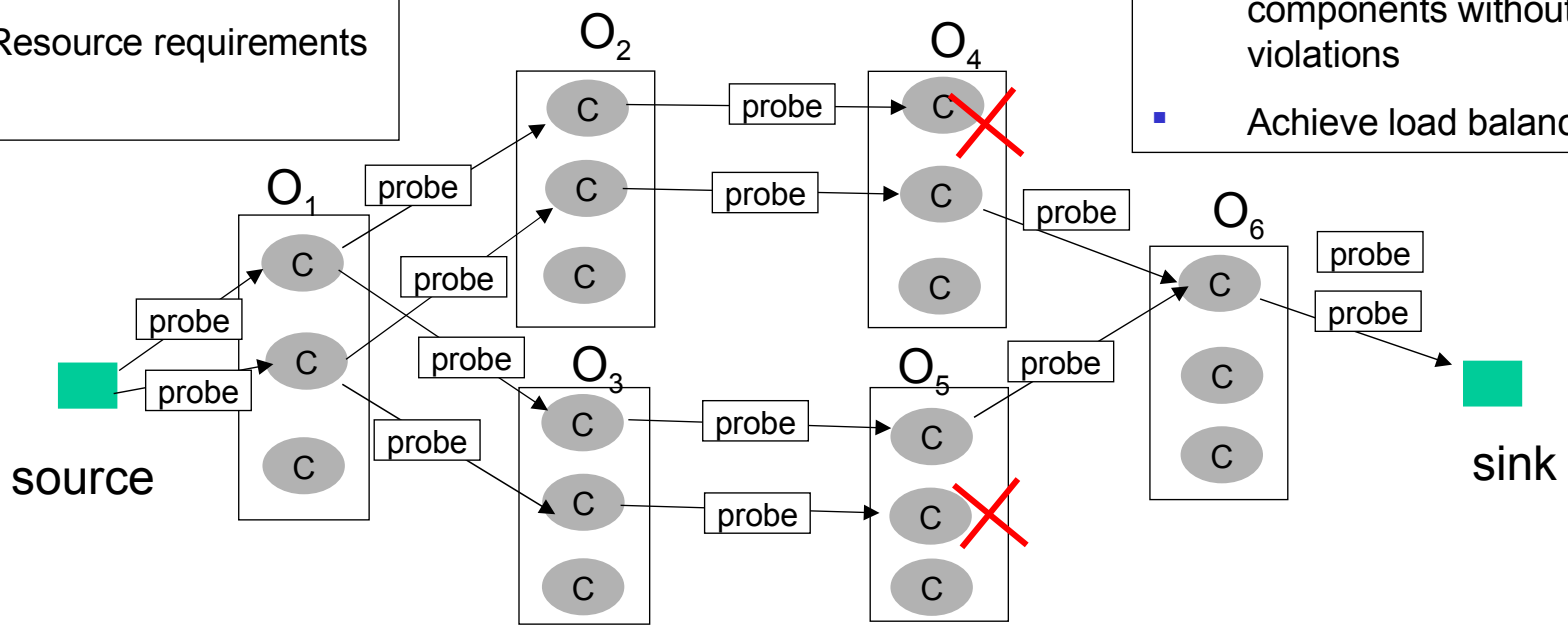
Query Plan

- 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



Processing a Probe

- Each node receiving a probe determines whether:
  - Resource requirements are satisfied
  - Requested QoS requirements are not violated
  - QoS of existing applications are not violated
- If conditions are met:
  - Resources are allocated transiently
  - Probes for next component are spawned

Composition Selection

- All successful probes return to source and are checked against constraints on:
  - operator functions
  - QoS
  - processing capacity
  - bandwidth
- The most load balanced one is selected among all qualified compositions:

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

Component Sharing

QoS Impact Projection algorithm

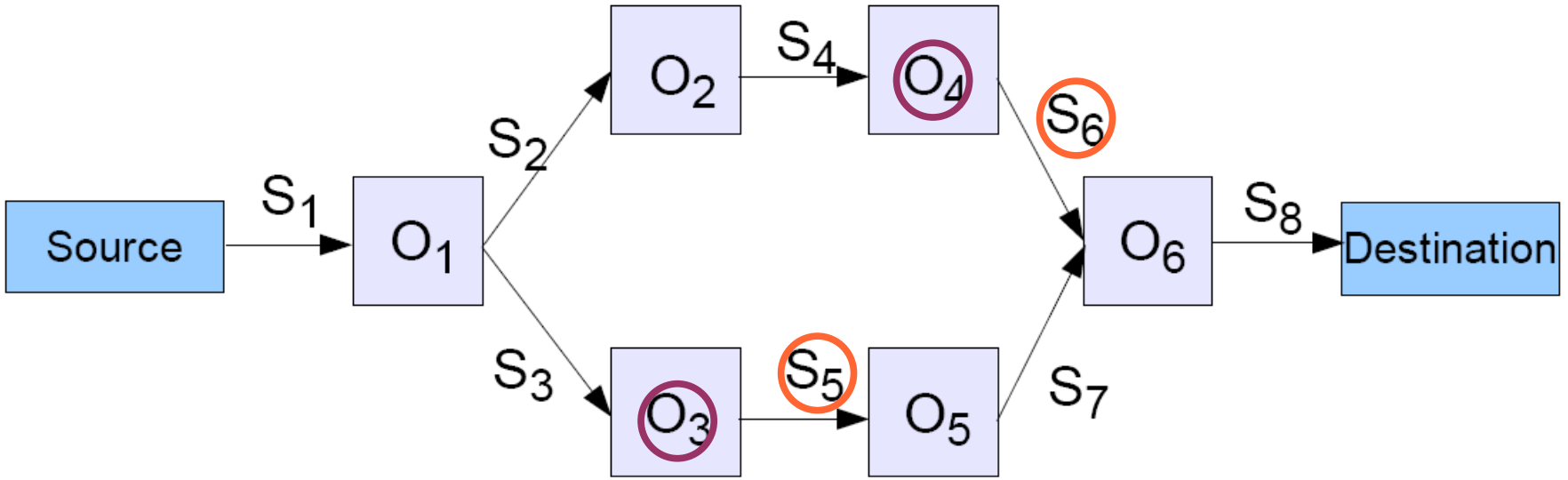
- All existing and the new application should not exceed requested execution time:

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

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

$$\delta = \hat{t}' - \hat{t} = \frac{\tau_{c_i}}{1 - (p_{v_i} + \tau_{c_i})} - \frac{\tau_{c_i}}{1 - p_{v_i}}$$

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

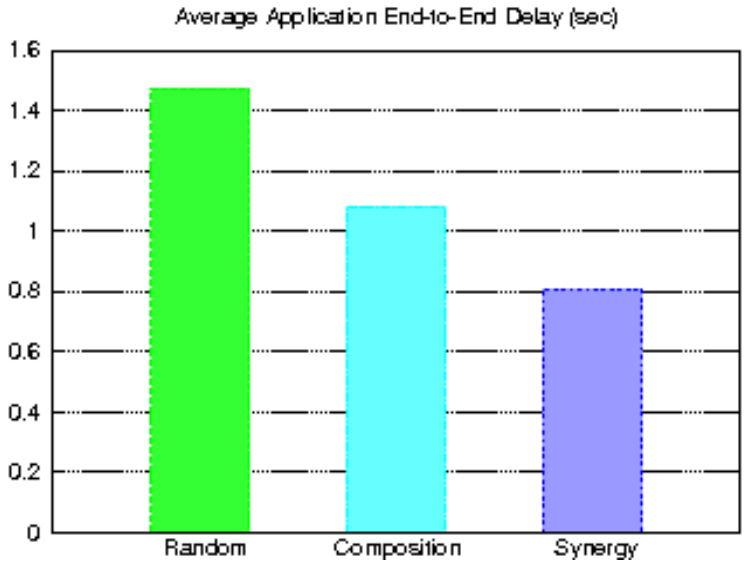
Roadmap

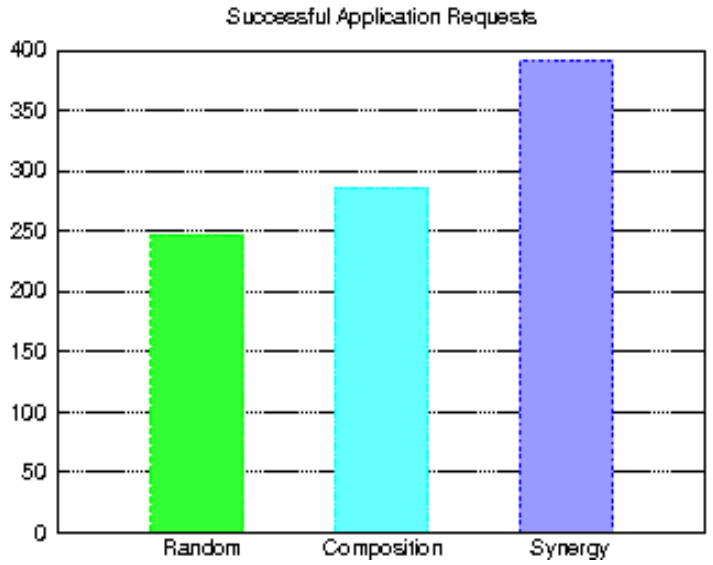
- Motivation and Background
- Synergy Architecture
- Design and Algorithms
  - Composition Protocol
  - Component Sharing
  - Stream Sharing
- Experimental Evaluation
- Related Work
- Conclusion

Experimental Evaluation

- PlanetLab multi-threaded prototype of about 18000 lines of Java running on 88 physical nodes
- Simulator of about 7500 lines of C++ for 500 random nodes of a GT-ITM topology of 1500 routers
- 5 replicas of each component
- Workload following Zipfian distribution
- Synergy vs Random, Greedy, and Composition

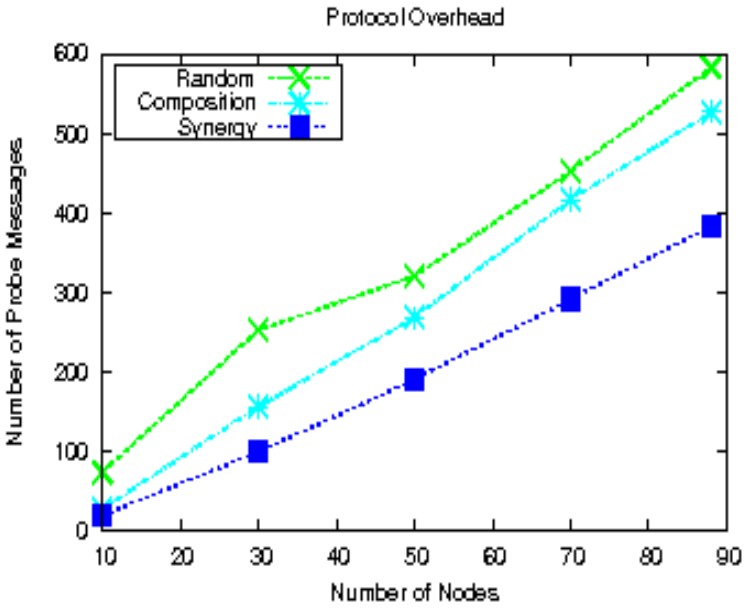
Performance on PlanetLab

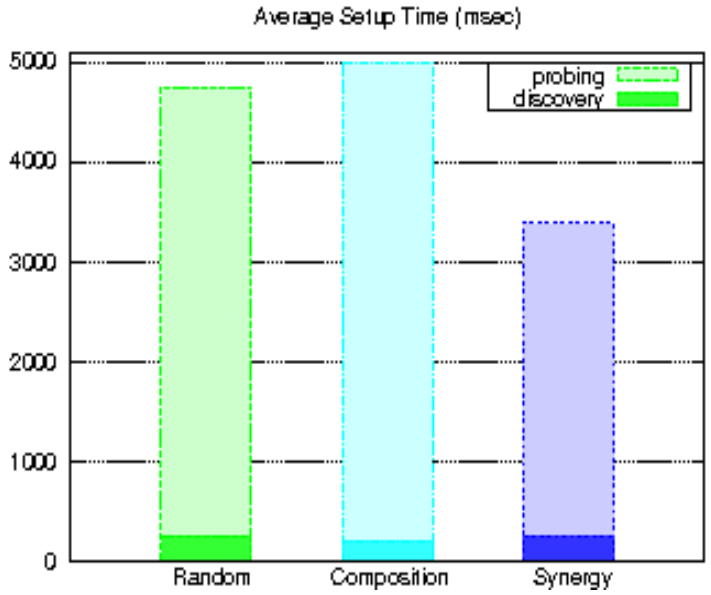




Stream reuse improves end-to-end delay by saving processing time and increases system capacity

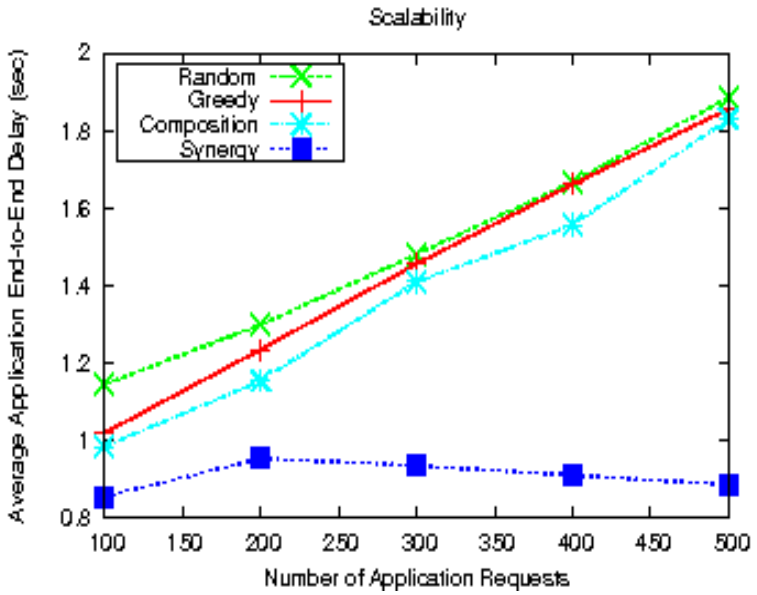
Overhead on PlanetLab

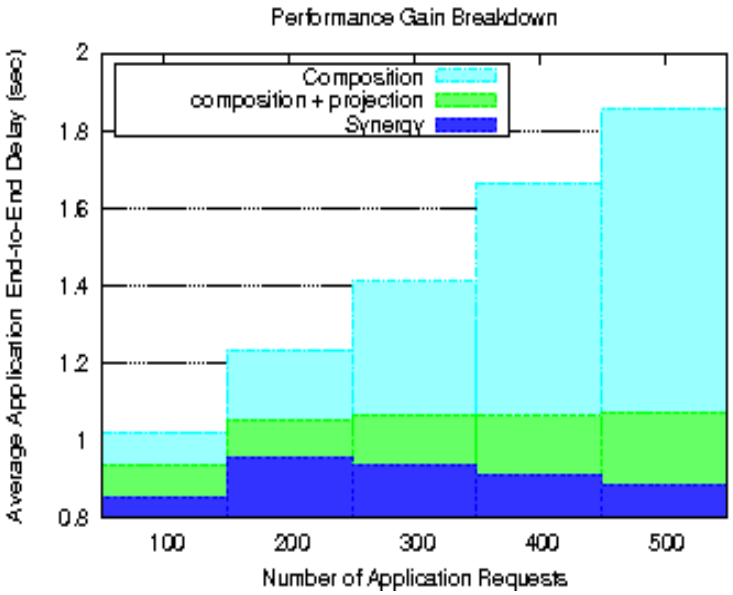




Stream reuse decreases probing overhead and setup time

Performance on Simulator





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

Related Work

- Distributed stream processing
  - *System S*: IBM stream processing middleware infrastructure
  - *SBON, SAND, IFLOW*: Component placement
  - *Borealis, Flux, PeerCQ*: Load balancing
  - *SpiderNet, sFlow*: Component composition
- Component composition
  - *BPEL, WS-Coordination, Adapt*: Web service composition
  - *ACE, QuO, AQuA*: QoS for DRE systems

Conclusion

- **Synergy: A distributed stream processing middleware providing sharing-aware component composition**
  - Overlay-based distributed stream processing
  - Fully distributed composition protocol
  - Maximum sharing discovery to reuse existing streams
  - QoS impact projection to reuse existing components
- **Future work**
  - Dynamic component migration
  - Failure resilience
- **Thank You!**
  -  http://www.cs.ucr.edu/~trep/

Thank You!

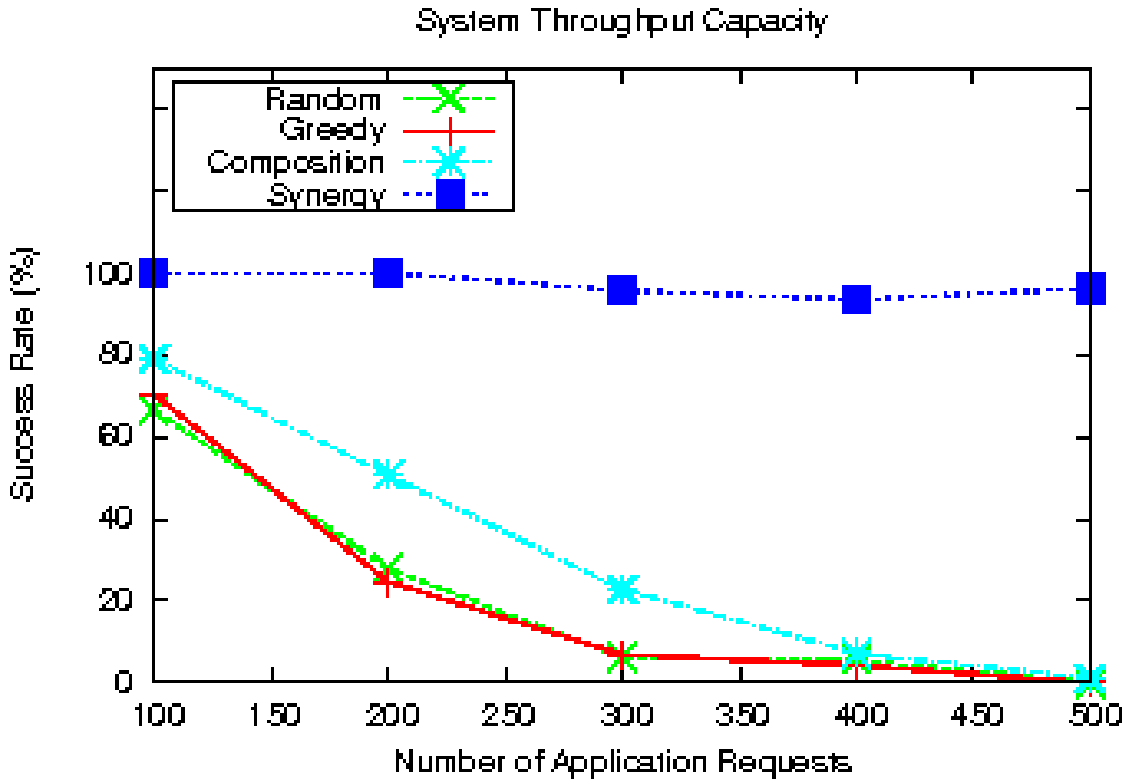
http://www.cs.ucr.edu/~trep/

Synergy: Sharing-Aware Component Composition for Distributed Stream Processing Systems

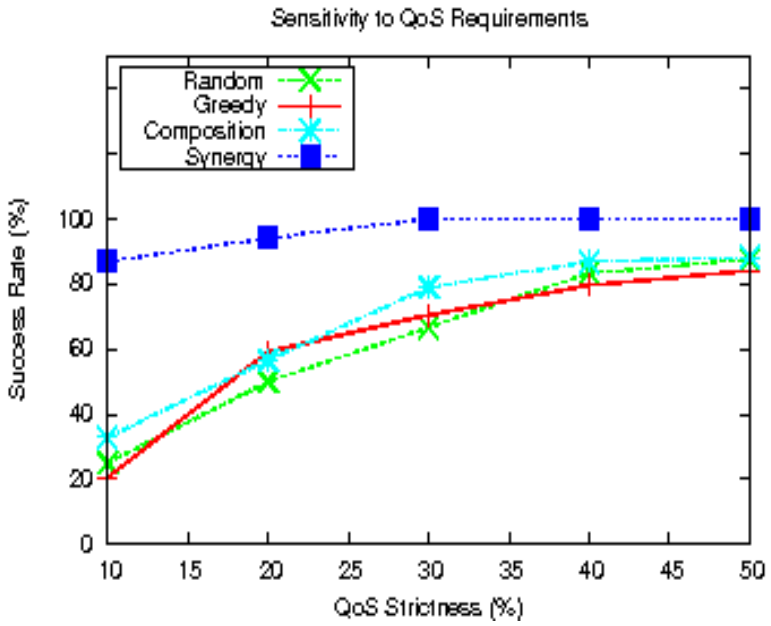
Thomas Repantis[1], *Xiaohui Gu*[2], *Vana Kalogeraki*[1]

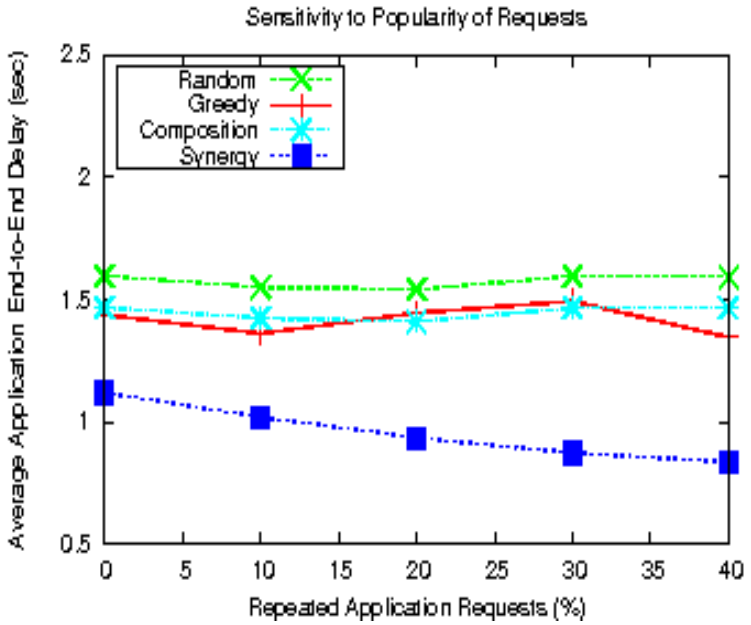
trep@cs.ucr.edu, xiaohui@us.ibm.com, vana@cs.ucr.edu
[1] University of California, Riverside
[2] IBM T.J. Watson Research Center, New York

Performance on Simulator



Sensitivity on Simulator





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

Sensitivity on Simulator