Accommodating Bursts in Distributed Stream Processing Systems

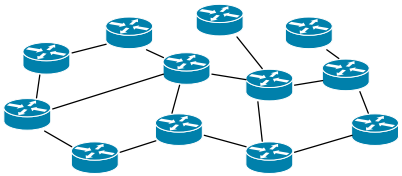
Yannis Drougas, Vana Kalogeraki
Distributed Real-time Systems Lab
University of California, Riverside

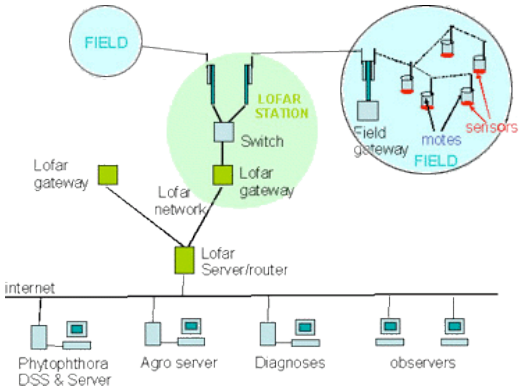
{drougas,vana}@cs.ucr.edu
http://www.cs.ucr.edu/~{drougas,vana}

Stream Processing Applications

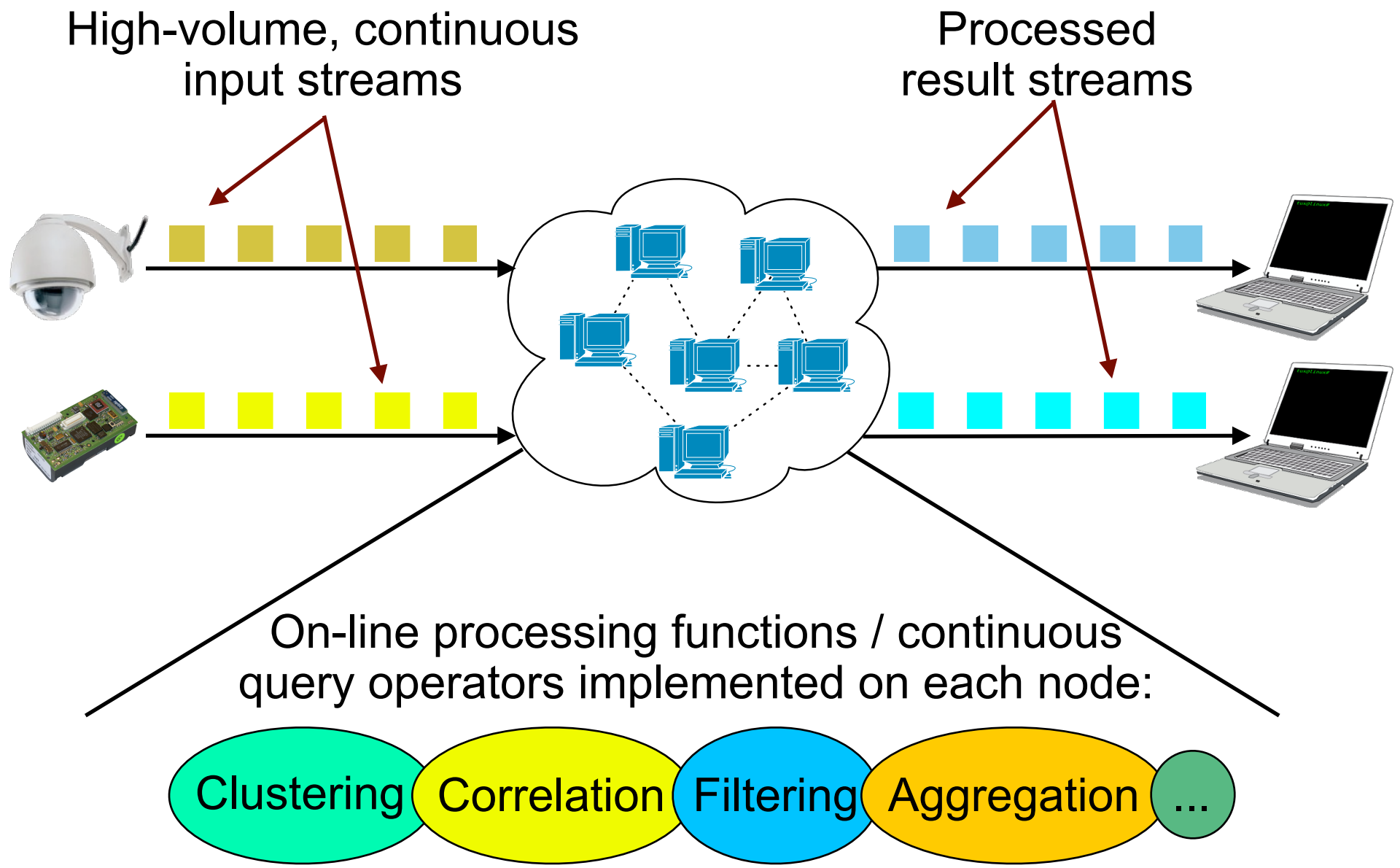
- Large class of emerging applications in which data streams must be processed online

- Example applications include:
  - Stock Exchange data filtering
  - Traffic Monitoring
  - Surveillance
  - Sensor network data processing
  - Network monitoring

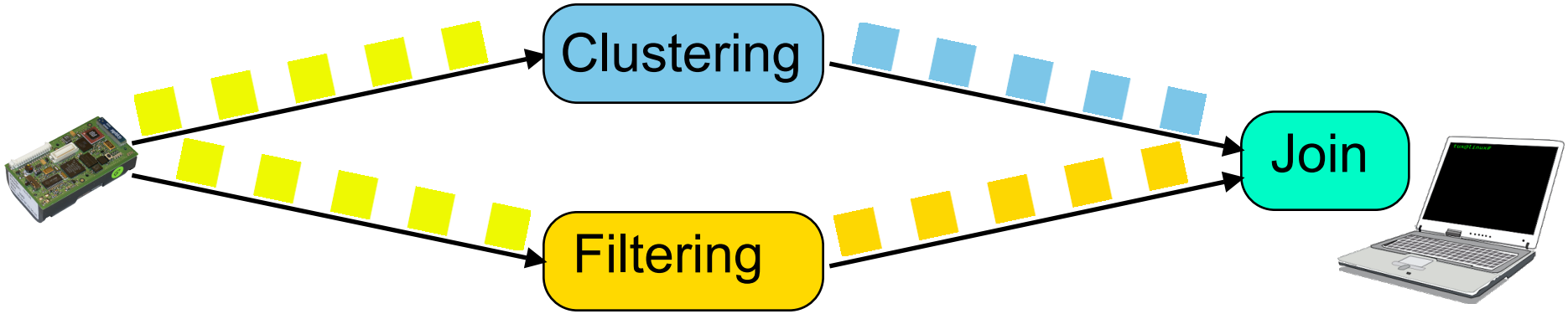




Distributed Stream Processing Systems



Stream Processing Applications Characteristics



- Data is produced continuously, in large volumes and at high rates
- Data has to be processed in a timely manner, e.g. within a deadline
- Application input rates fluctuate notably and abruptly

Previous Work

- The majority of previous work [FIT, MPRA, Brown@VLDB 2006, Amini@ICDCS 2006, Xia@ICDCS 2007] has focused on optimizing a given utility function
  - Some solutions [FIT] employ data admission
  - Others [MPRA, Brown@VLDB 2006] consider the optimal placement of tasks on nodes
- The case where load patterns can be predicted has also been studied [Borealis @ ICDE 2005]
- QoS management [RTStream] is another solution

Our Problem

- We focus on the problem of addressing bursts of input data rate
  - Devise a plan to thwart the burst
  - Provision for future bursts
- Benefits:
  - Lost data units due to bursts are minimized
  - No QoS degradation or data admission
  - No under-utilization, dynamic reservation used
- Challenges:
  - Highly dynamic / unpredictable environment
  - Multiple limiting resource types
  - Plan must be applied on time for the burst

Roadmap

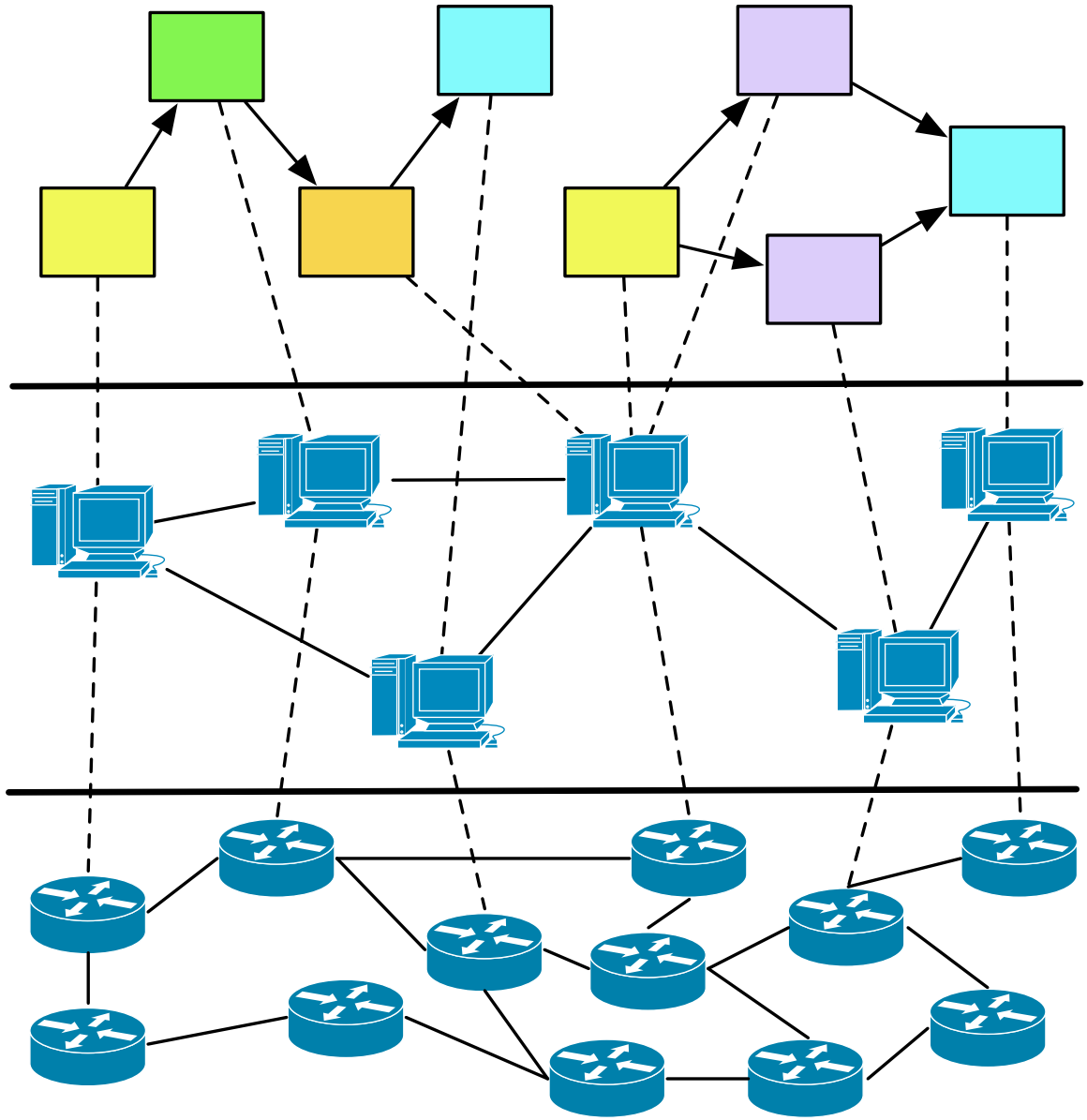
- Motivation and Background
- System Architecture
- Burst Handling Mechanism
  - Feasible Region & Index Points
  - Application-based Reservation
  - Online system adjustment
- Experimental Evaluation
- Conclusion

System Architecture

Application layer: Execution of stream processing applications.

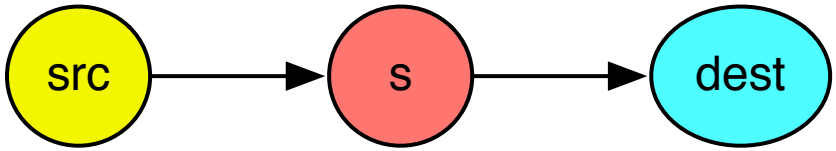
Overlay network consisted of processing nodes. Built over a DHT (currently, Pastry).

The physical (IP) network.

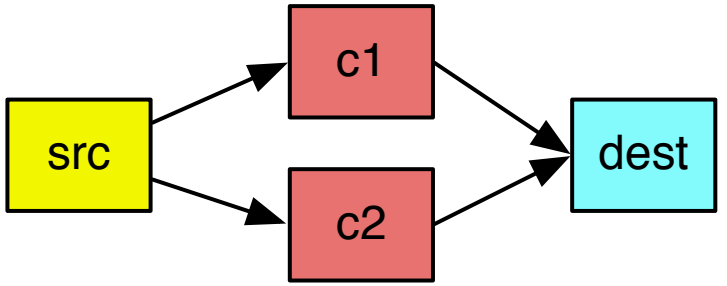


Application Execution

- A stream processing application is executed collaboratively by peers of the system that invoke the appropriate services.
- A service can be instantiated on more than one nodes.
- A service instantiation on a node is a component.

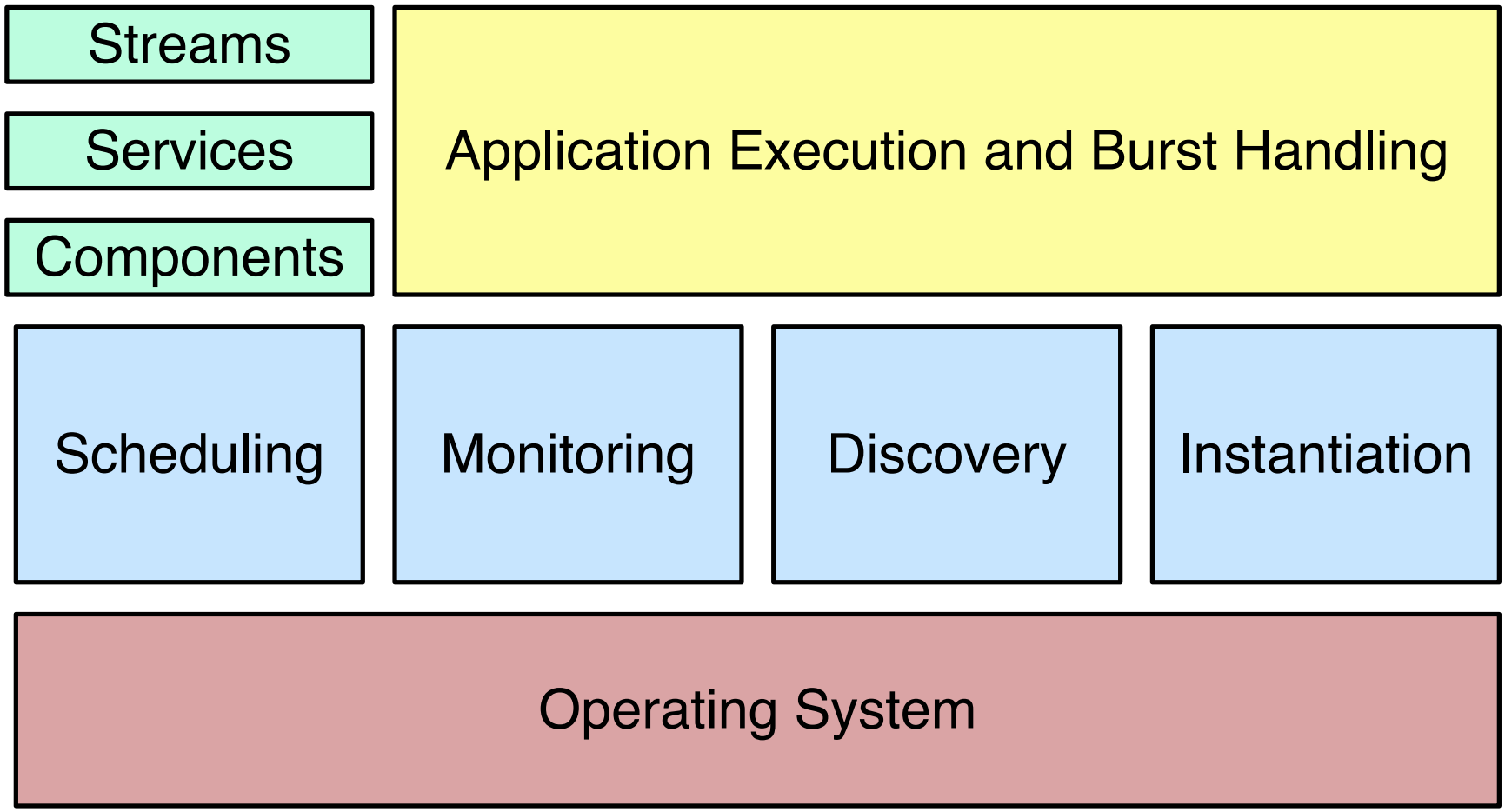


Application submitted by the user



Application executed on the system

System Architecture



System Model

- Each component is characterized by its resource requirements
- Selectivity is another component characteristic.

$$\mathbf{u}^{c_i} = \begin{bmatrix} u_1^{c_i} \\ u_2^{c_i} \\ \dots \\ u_J^{c_i} \end{bmatrix}$$

$$sel^{c_i} = \frac{average\ output\ rate}{average\ input\ rate}$$

- Each node is characterized by the availability of its resources.

$$\mathbf{A}^n = \begin{bmatrix} A_1^n \\ A_2^n \\ \dots \\ A_J^n \end{bmatrix}$$

Roadmap

- Motivation and Background
- System Architecture
- Burst Handling Mechanism
  - Feasible Region & Index Points
  - Application-based Reservation
  - Online system adjustment
- Experimental Evaluation
- Conclusion

Optimization Problem

- Capacity Constraints:

$$\forall n \in \mathcal{N}, \sum_{c_i \in n} r_{c_i} \cdot u_j^{c_i} \leq A_j^n, 1 \leq j \leq J$$

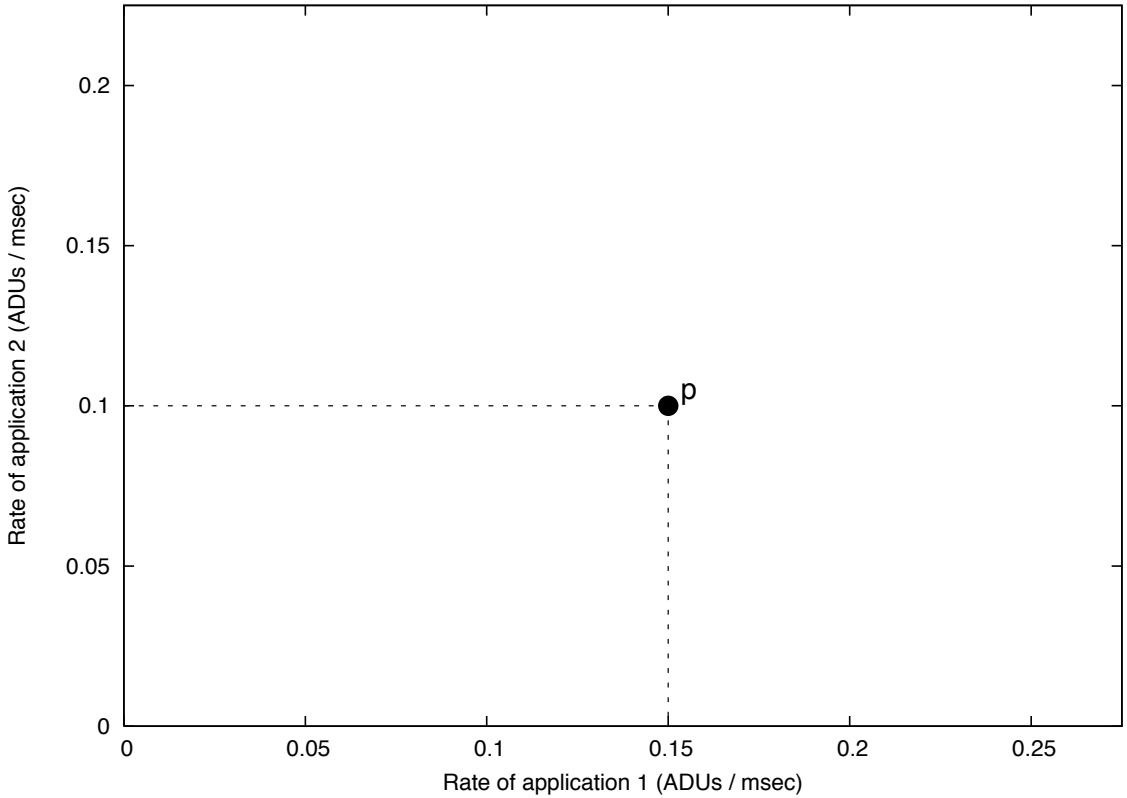
- Flow Conservation Constraints:

$$\forall c_i, \sum_{c_j \in \mathcal{D}(c_i)} r_{c_j} = sel^{c_i} \cdot r_{c_i}$$

- We need to come up with a plan that satisfies the above constraints and minimizes the likelihood of missing data due to bursts.

Feasible Region

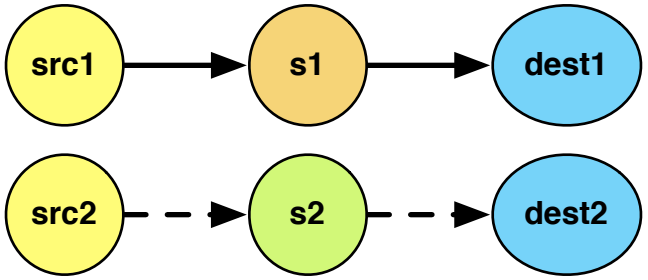
- Assume we have Q applications
- The state of the system at any given time can be described by a point in the Q-dimensional space



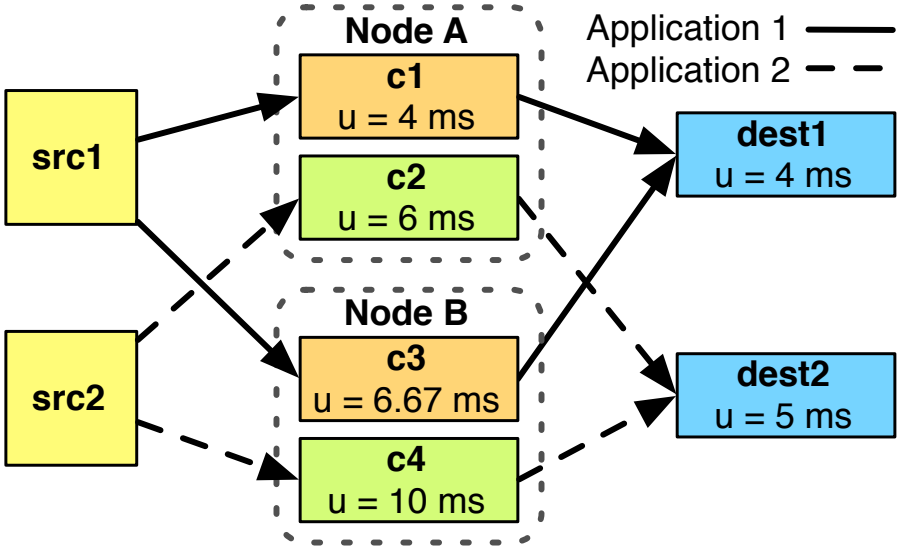
Feasible Region

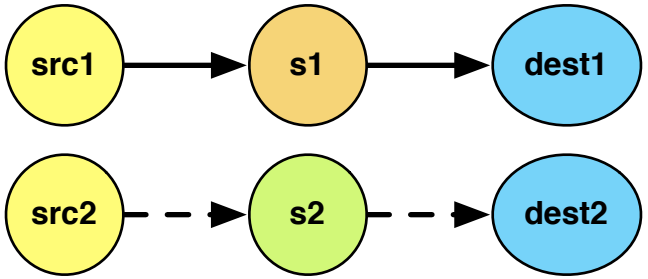
- The feasible region is the set of all points (application input rates combinations) that nodes in the given distributed stream processing system can accommodate without any data unit being dropped.
- The form of linear constraints suggest that in the general case of Q applications, the feasible region is a convex polytope.

Feasible Region - Example

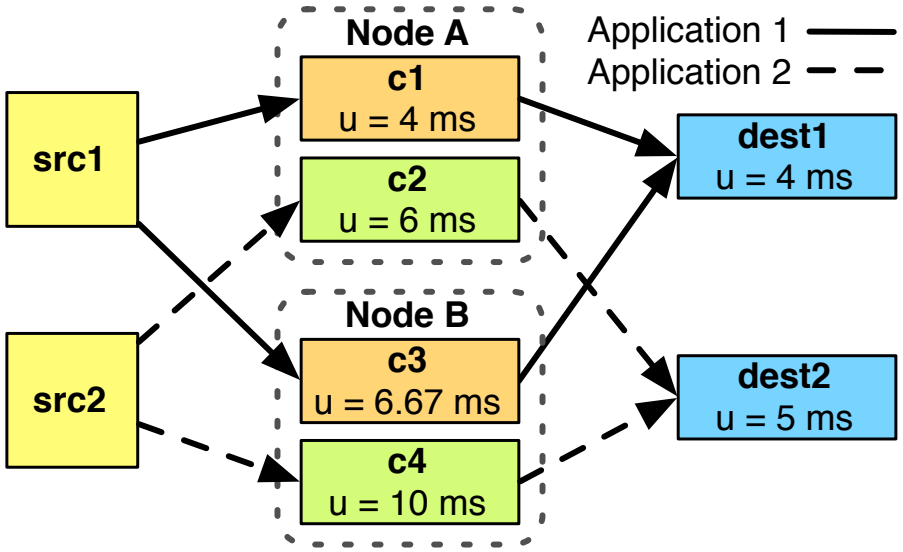


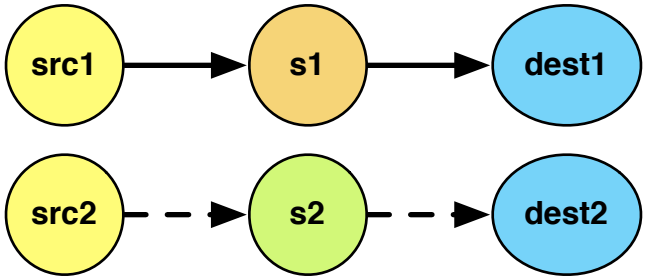
Feasible Region - Example





Feasible Region - Example





$$r_{c_1} \cdot 4 + r_{c_2} \cdot 6 \leq 1$$

$$r_{c_3} \cdot 6.67 + r_{c_4} \cdot 10 \leq 1$$

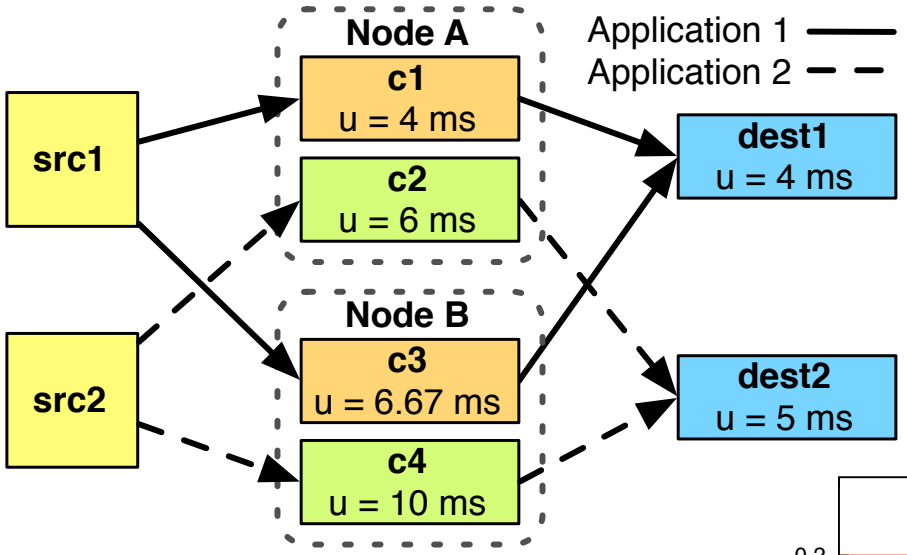
$$r_{dest_1} \cdot 4 \leq 1$$

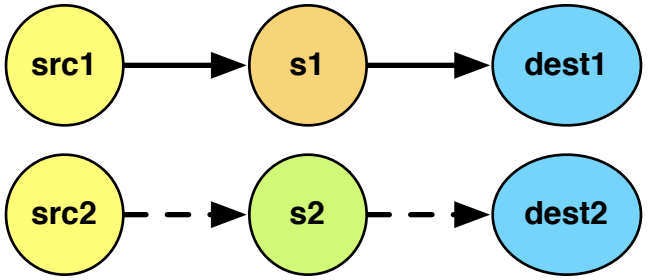
$$r_{dest_2} \cdot 5 \leq 1$$

$$r_{dest_1} = r_{c_1} + r_{c_3}$$

$$r_{dest_2} = r_{c_2} + r_{c_4}$$

Feasible Region - Example





$$r_{c_1} \cdot 4 + r_{c_2} \cdot 6 \leq 1$$

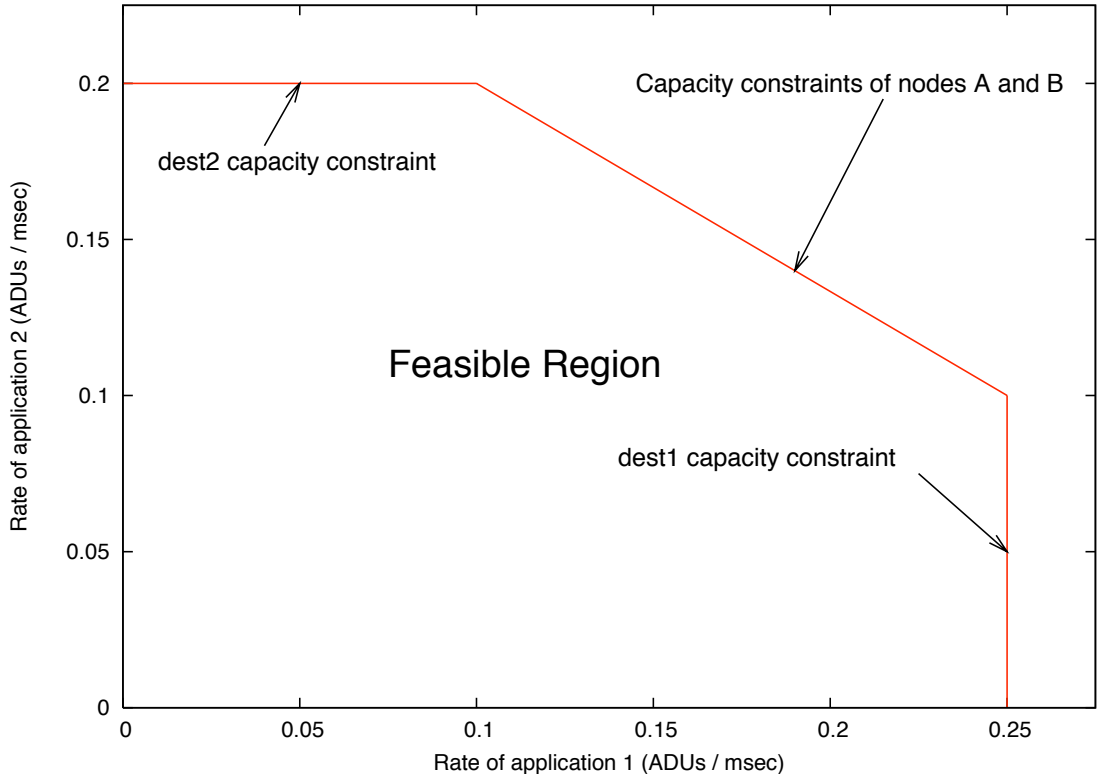
$$r_{c_3} \cdot 6.67 + r_{c_4} \cdot 10 \leq 1$$

$$r_{dest_1} \cdot 4 \leq 1$$

$$r_{dest_2} \cdot 5 \leq 1$$

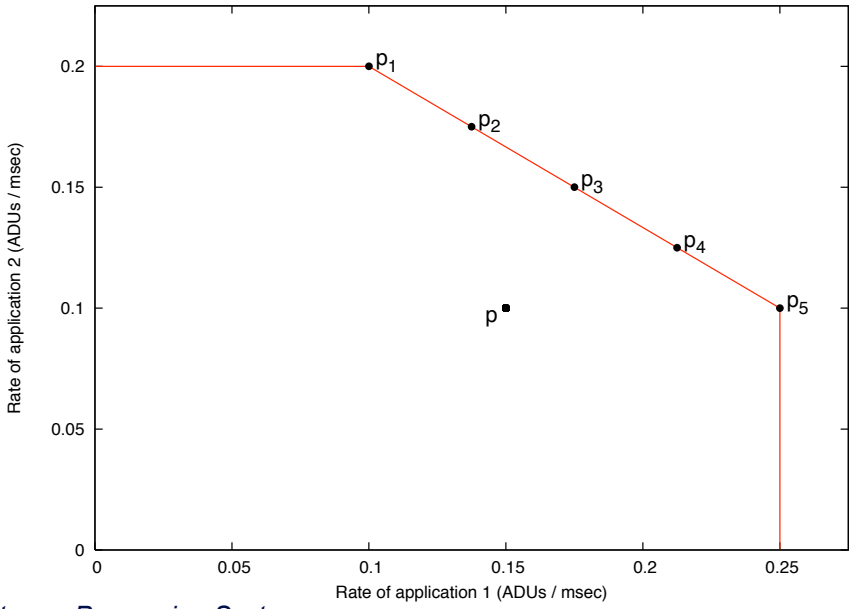
$$r_{dest_1} = r_{c_1} + r_{c_3}$$

$$r_{dest_2} = r_{c_2} + r_{c_4}$$



Dominance & Pareto Points

- A point p1 dominates a point p2 when for each application q, $p_1(q) \geq p_2(q)$
- If the current system state is p2, one can apply the rate allocations calculated for p1
- A Pareto point is not dominated by any other point in the feasible region
- Pareto points represent optimal solutions: There is no point that is "better" than a pareto point



Burst Handling

- If the input rate of application q increases by δ_q, the input rate of a component c_i of q will increase by $\delta_q \cdot \frac{r_{c_i}}{r_q}$

- In order for a stream processing system to be able to sustain such an increase, the following must hold for each node:

$$\underbrace{\sum_{c_i \in n} r_{c_i} \cdot u_j^{c_i}}_{\text{Initial resource requirements}} + \underbrace{\sum_{c_i \in n \cap \mathcal{C}^q} \delta_q \cdot \frac{r_{c_i}}{r_q} \cdot u_j^{c_i}}_{\text{Additional resource requirements due to single burst}} \leq A_j^n$$

Optimization Objective

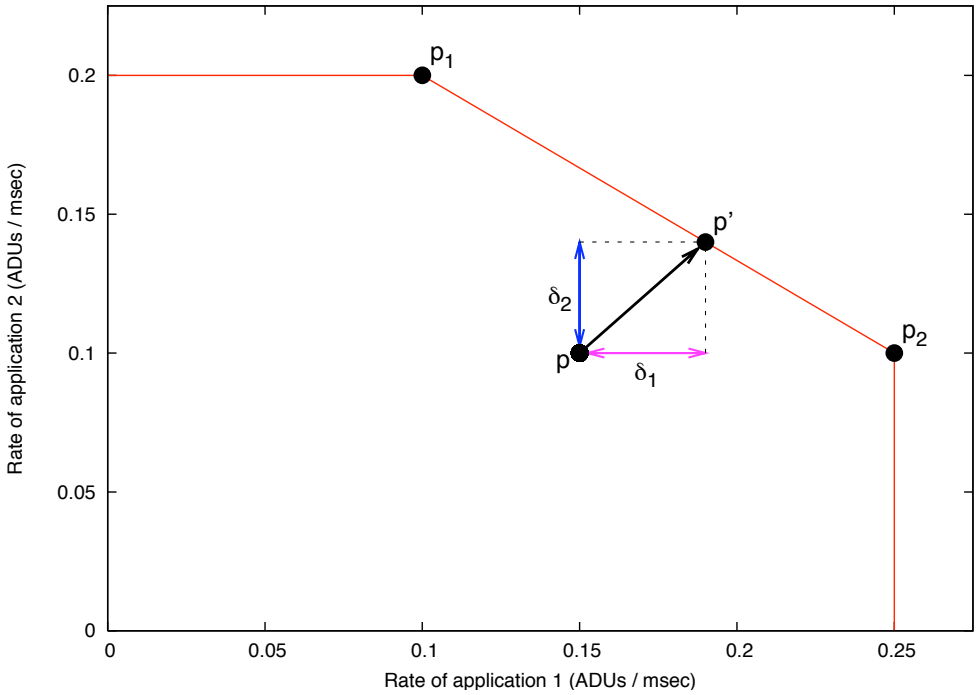
- To minimize the amount of dropped data, we wish to maximize $\sum \delta_q$
- If the current input rates are represented by p, we need to configure the system for:

$$p' = p + \boldsymbol{\delta} = \begin{bmatrix} r_1 + \delta_1 \\ \cdots \\ r_Q + \delta_q \end{bmatrix}$$

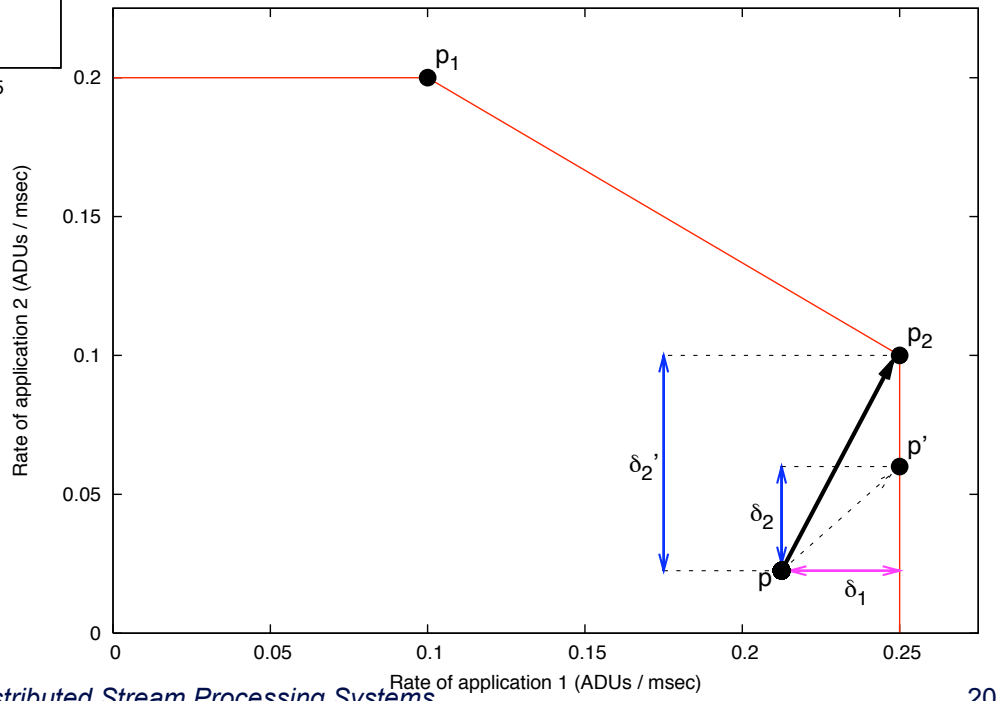
- We assume each application has equal probability for a burst to appear. So, δ_q's must be as equal as possible:

$$p' \succcurlyeq \begin{bmatrix} r_1 + c \\ \cdots \\ r_Q + c \end{bmatrix} \Rightarrow \boldsymbol{\delta} = \begin{bmatrix} \delta_1 \\ \cdots \\ \delta_Q \end{bmatrix} \succcurlyeq \begin{bmatrix} c \\ \cdots \\ c \end{bmatrix}$$

Optimization Objective - Example



The optimal point p' is the one for which $\delta_1 = \delta_2$



The optimal point is p2, since

$\delta_2' > \delta_2 \Rightarrow p_2 \succcurlyeq p'$

BARRE

- Incorporating bursts makes capacity constraints non-linear.
- Re-calculating the optimal component rate assignment when a burst appears would be too slow.
- Instead, our solution:
  - Pre-calculates a small number of component rate assignment plans during an offline phase.
  - Monitors application incoming rates and resource availability during runtime.
  - When bursts occur, component rates are assigned, based on the pre-calculated plans.

Feasible Region Determination

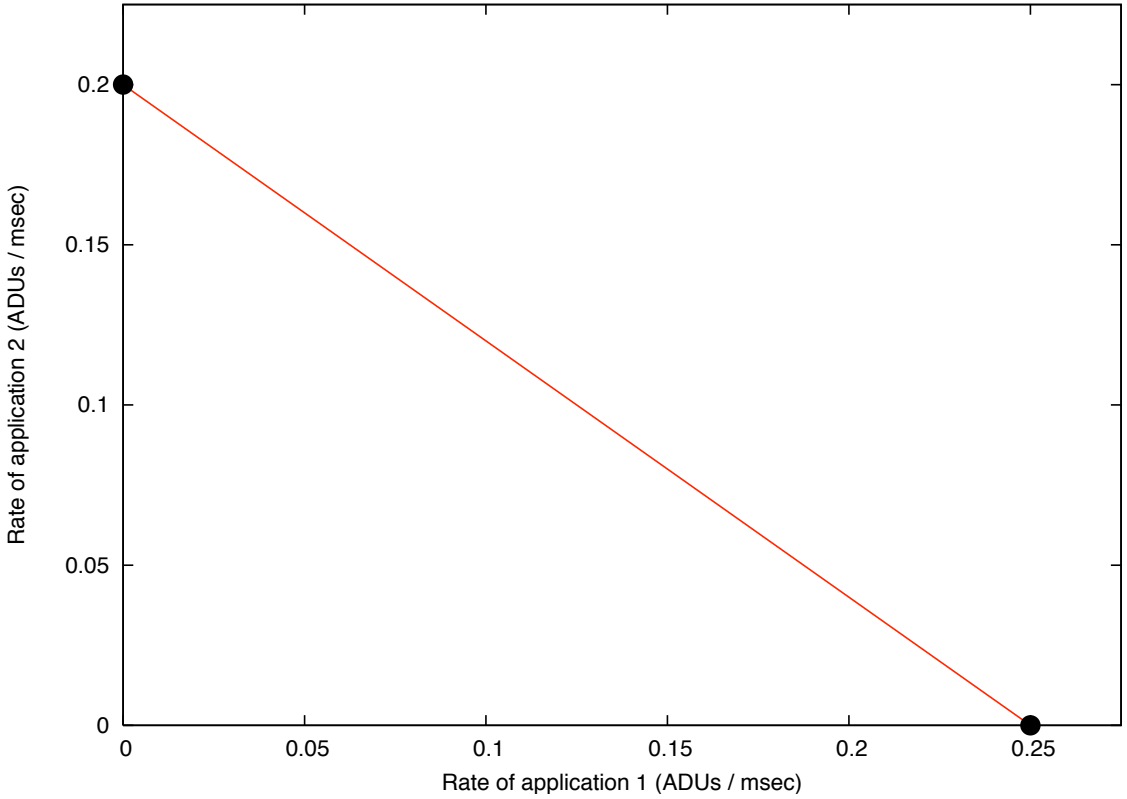
- We need to select some index points and pre-calculate their optimal rate assignment
- Index points are a subset of Pareto points: They are the “vertices” of the feasible region.
- These component rate assignments will be used to construct the appropriate component rate allocation on the event of a burst.
- For each index point, a list of the feasible region’s sides that are adjacent to the index point, is kept.

Identifying the Index Points

- Find the maximum rate for each application
  - By solving a max-flow problem, under the capacity and flow conservation constraints
- Consider the resulting points as the vertices of a side with (Q-1) dimensions
  - This is since the Q-th dimension is a linear combination of the others
- Find the mid-point p_0 of the side, as well as the normal (perpendicular) vector d_0
- Move p_0 to the direction of d_0 as long as constraints are satisfied
- Repeat as long mid points can be moved

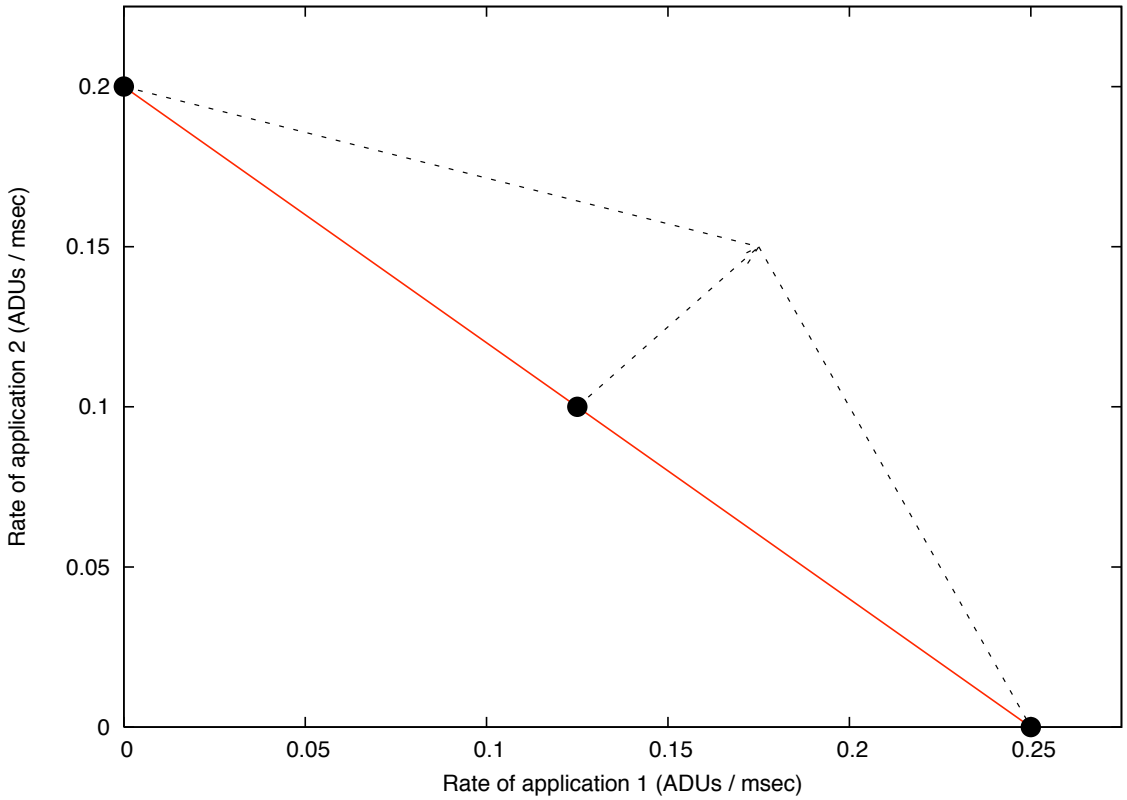
Identifying Index Points - Example

- Step 1: Find the maximum possible rate for each application
  - Solve a max-flow problem, with the rates of all but one applications set to 0.



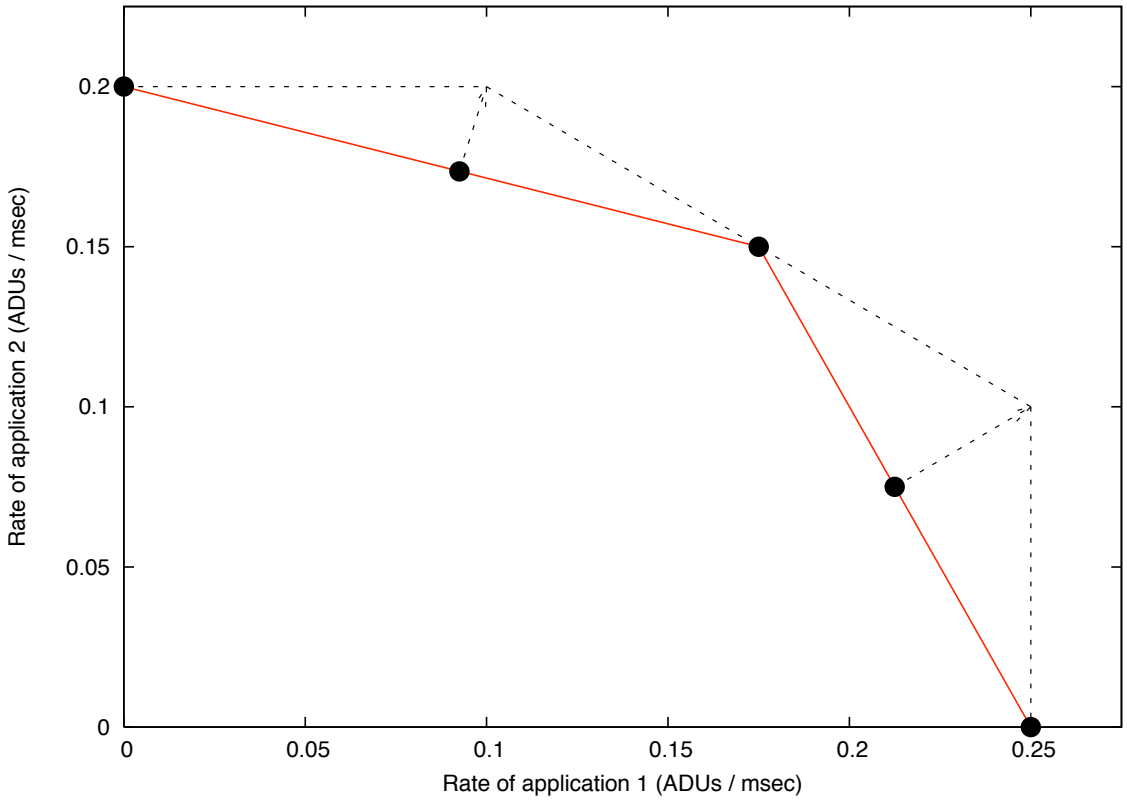
Identifying Index Points - Example

- Step 2: Find the mid-point of the resulting plane and see how far it can go
  - Solve max-flow by also constraining direction



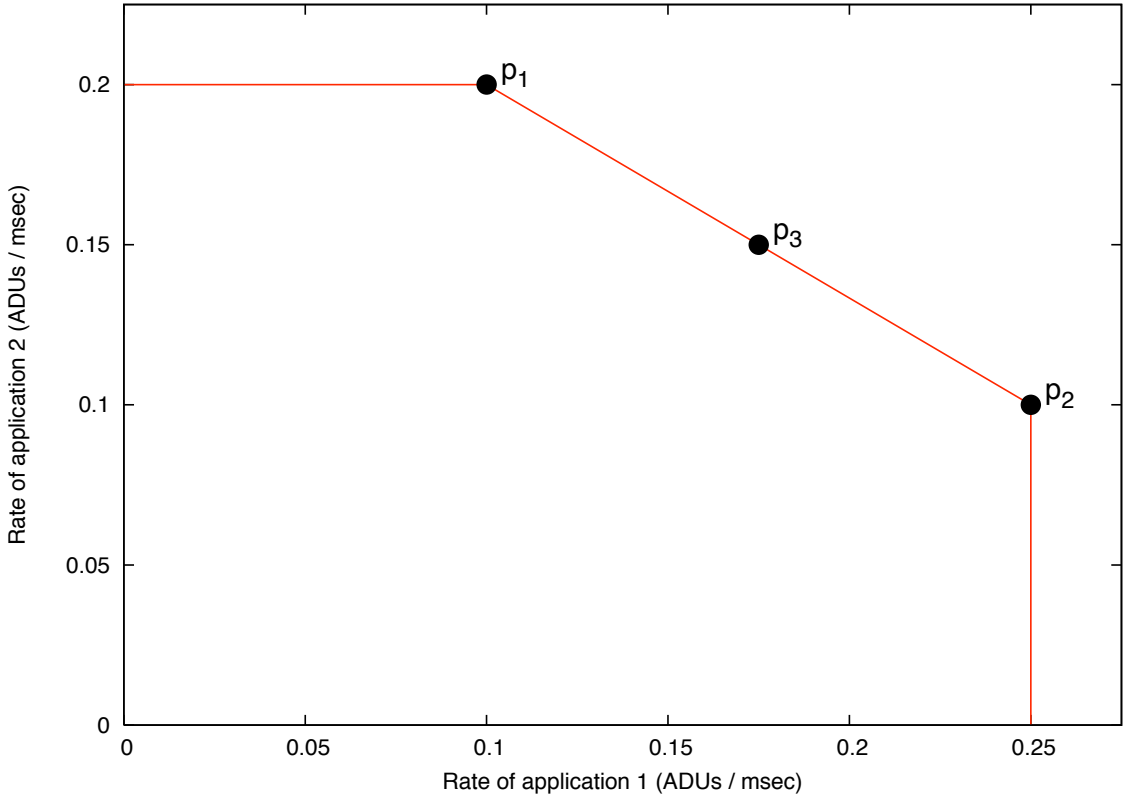
Identifying Index Points - Example

- Step 3: For each of the resulting sides, find its mid-point and repeat previous step
  - The upper left and lower right points are now dominated by the newly found index points



Identifying Index Points - Example

- Step 4: This is the resulting feasible region
  - The mid-points of the sides cannot improve without breaking the capacity constraints
- We end up with triplets: $< p, sol(p), \mathcal{F}(p) >$

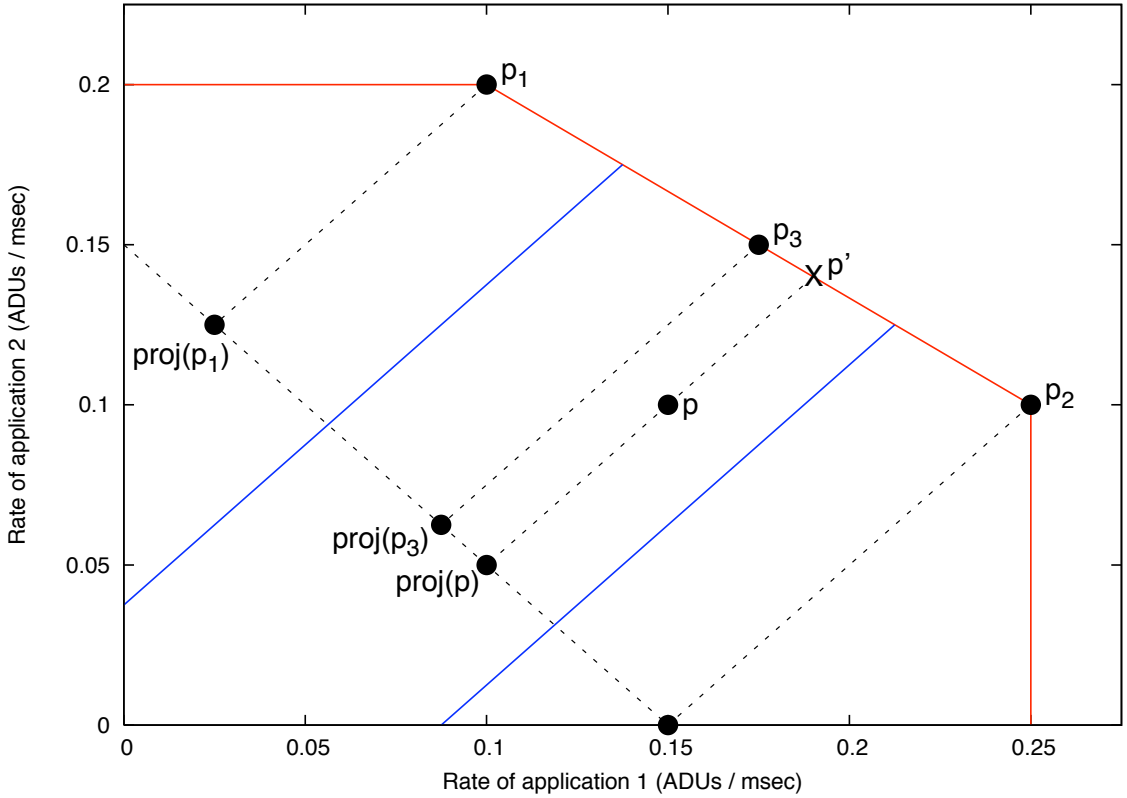


Storing Index Points

- Index points implicitly split the feasible region into subregions
  - For each subregion, there is one index point p
  - That index point is the closest to the optimal solution for points (application input rate combinations) in the subregion
- To identify the appropriate index point for any given application input rate, all points are projected on plane [1,1, ..., 1]
  - This way, the implicit split of the feasible region is brought out

Storing Index Points

- The closest to proj(p) the projection of p3 is, the closest to the optimal solution.
  - Search utilizes an R-tree indexed by the projections of the index points



Online Component Rate Assignment

- Upon a burst, the new application input rates are represented by a point p. We determine the appropriate component input rate allocation plan for p, that will maximize $\sum \delta_q$, while respecting the constraints
  - There is a pareto point p', the optimal solution of which is the same as the one for p
  - Any pareto point can be expressed as a linear combination of (at most) Q index points
  - The optimal solution of a pareto point is thus a linear combination of the optimal solutions of (at most) Q index points

Rate Assignment Algorithm

- Given the current system state p:
  - Find the index point p_i closest to p
  - If p_i is equal to p, return the solution pre-calculated for p_i
  - Otherwise, retrieve the sides of the feasible region for which p_i is a vertex.
  - The optimal point p' is a linear combination of the index points of one of these sides:

$$p' = a_1 \cdot p_1 + \ldots + a_Q \cdot p_Q, \quad 0 \leq a_q \leq 1, \quad \sum_{q=1}^{Q} a_q = 1$$

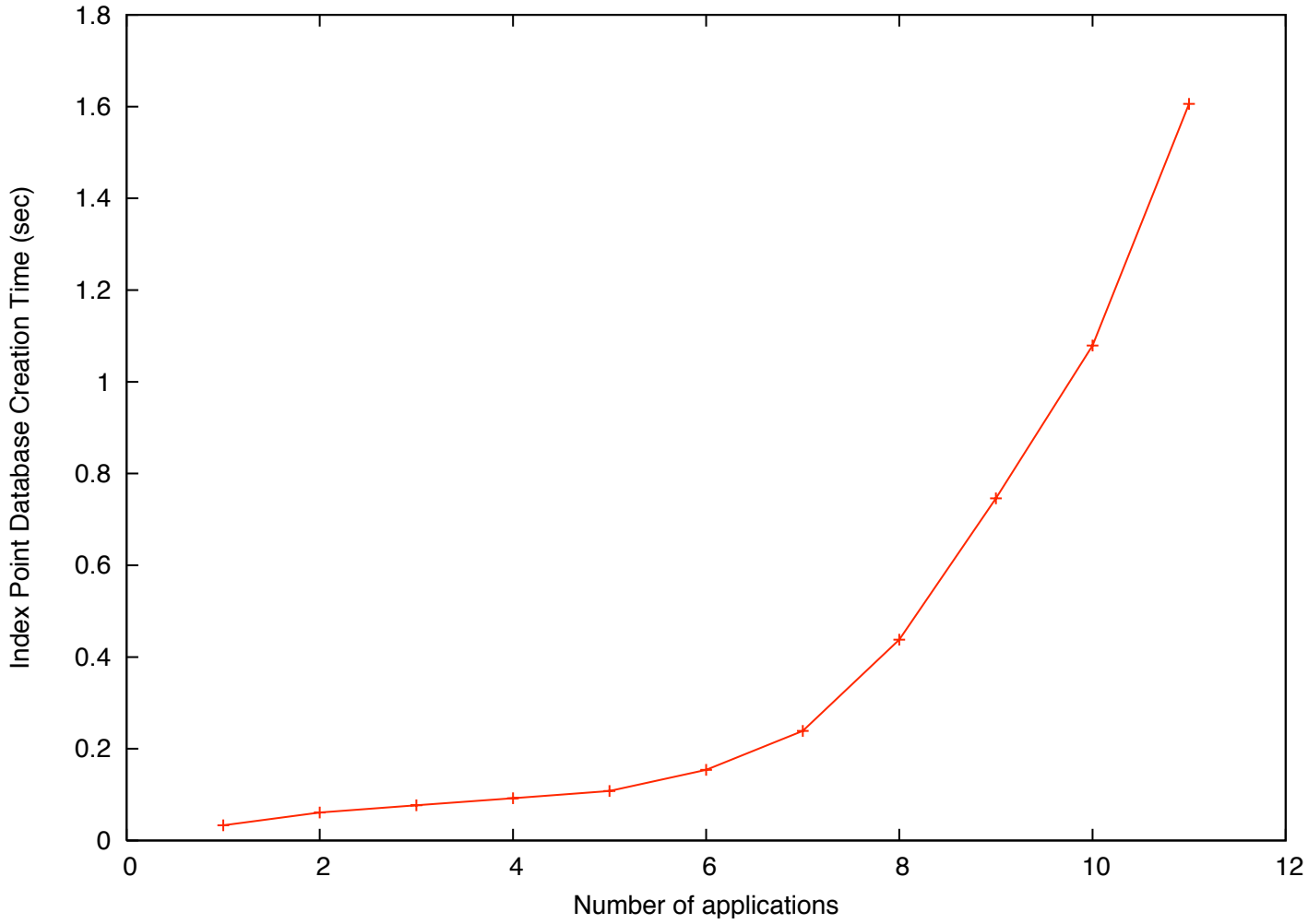
  - The solution is also a linear combination of their solutions:

$$sol(p) = sol(p') = a_1 \cdot sol(p_1) + \ldots + a_Q \cdot sol(p_Q)$$

Experimental Setup

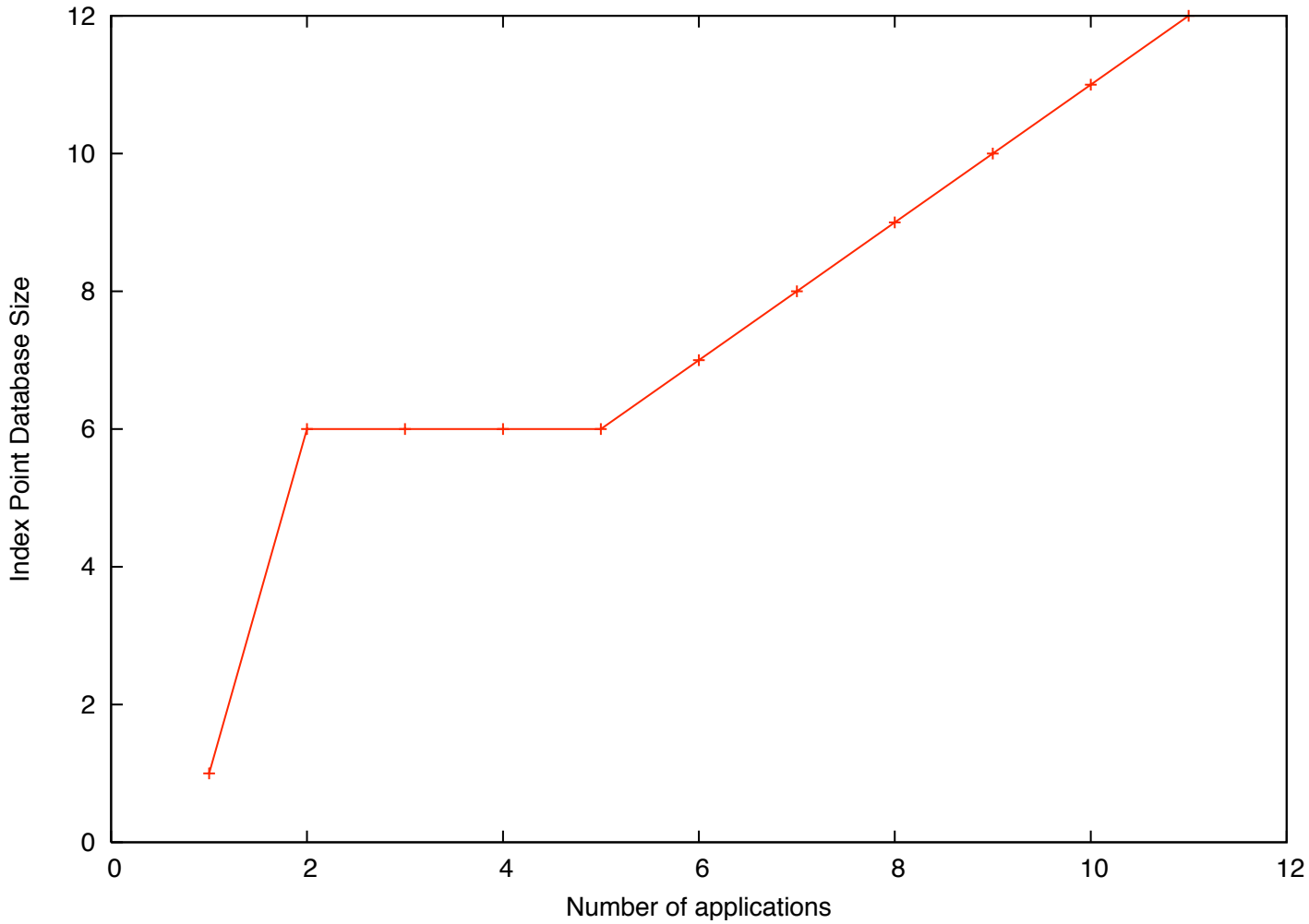
- Implementation over Synergy
- Used FreePastry for service discovery and collection of statistics
- 5 repetitions of each experiment
- 10 unique services in the system, 5 services on a single node
- Each application included 4 to 6 services
- Comparison with:
  - A burst-unaware method
  - Static Reservation of 20% of node resources
  - Simple Dynamic Adaptation, from previous work
  - A combination of the above

Index Point Pre-Calculation



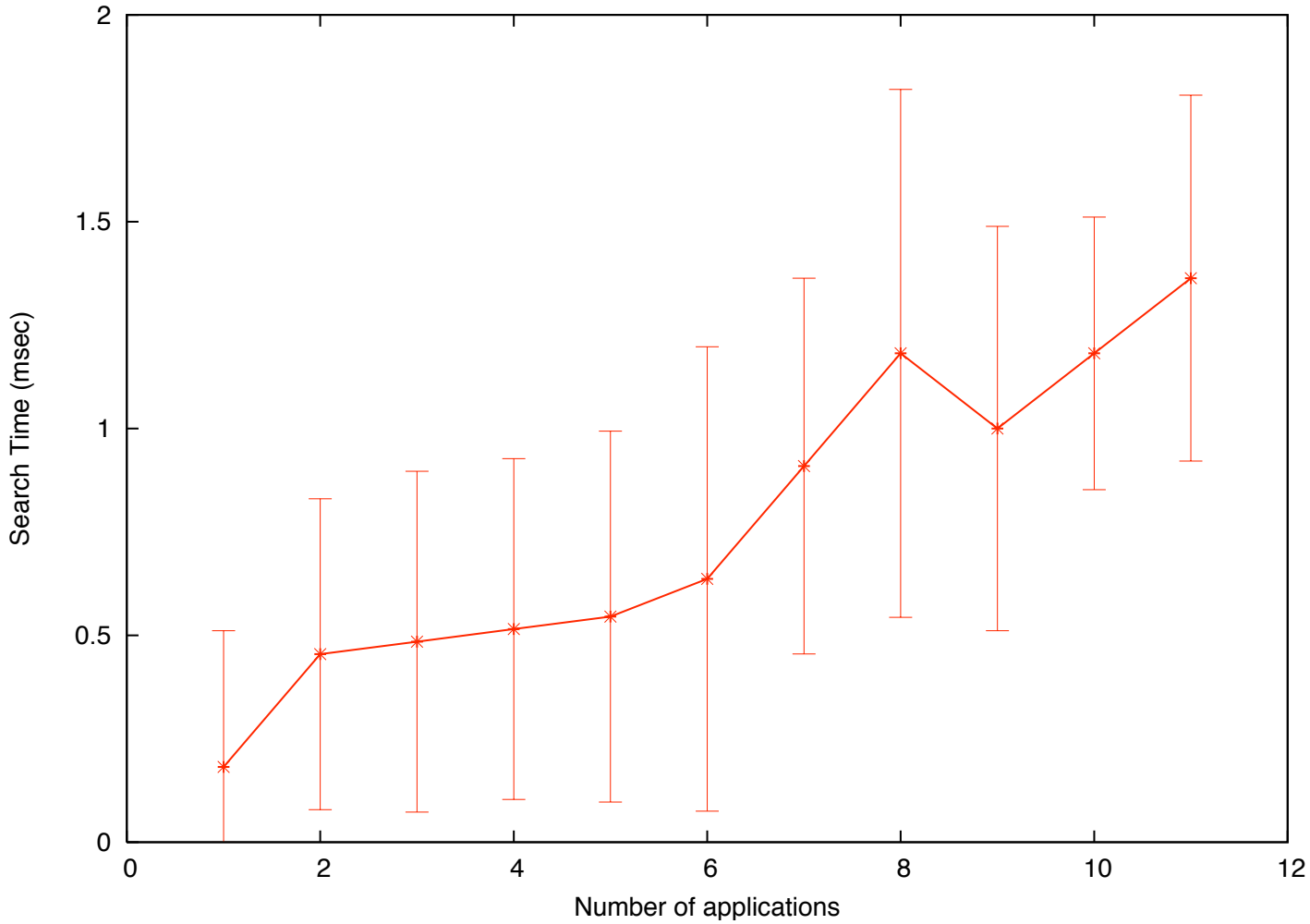
Exponential Index point calculation time, this is why index points need to be pre-calculated.

Index Point Database Size



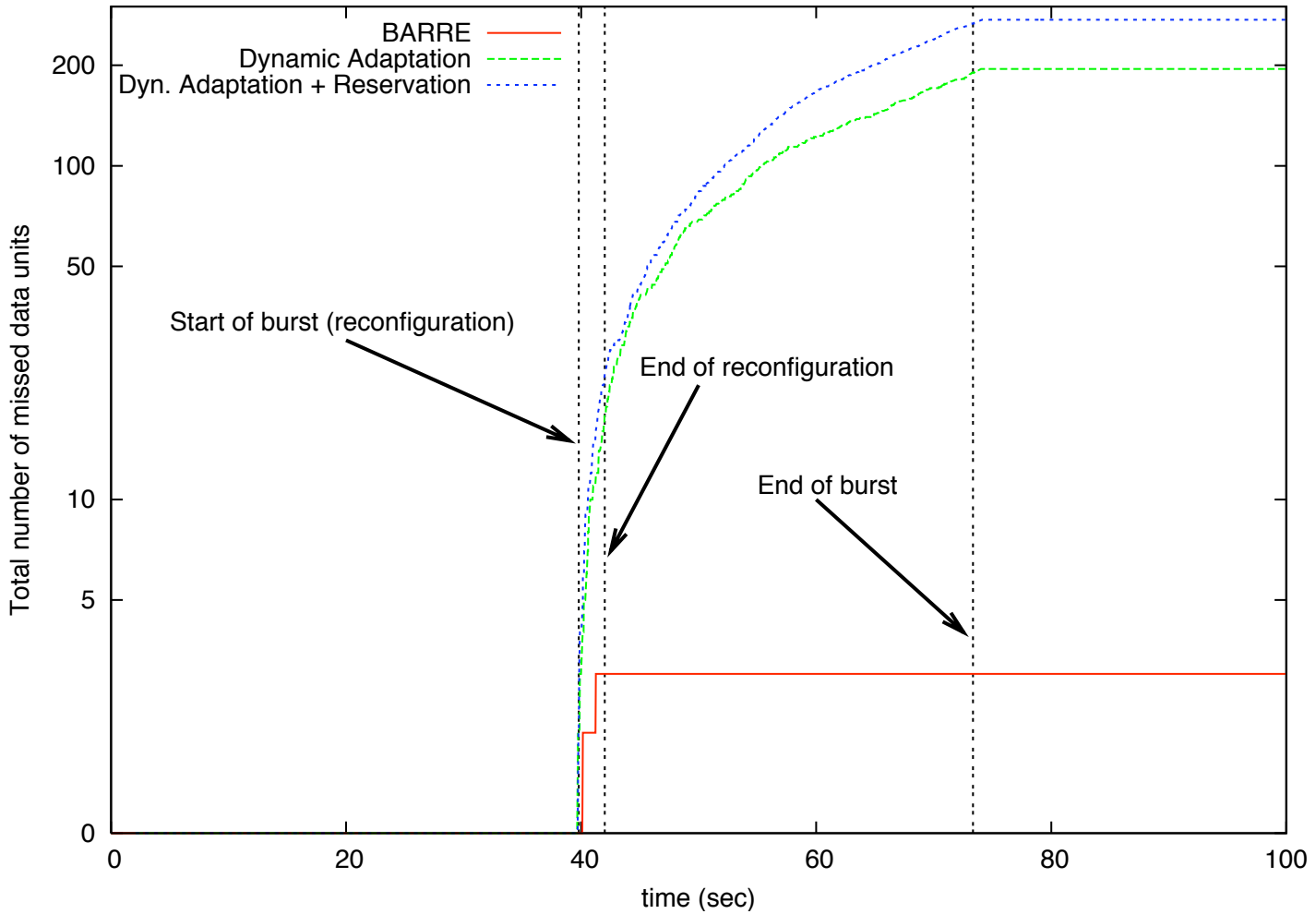
The size of the Index Point Database increases linearly with the number of applications.

Optimal Point Search Time



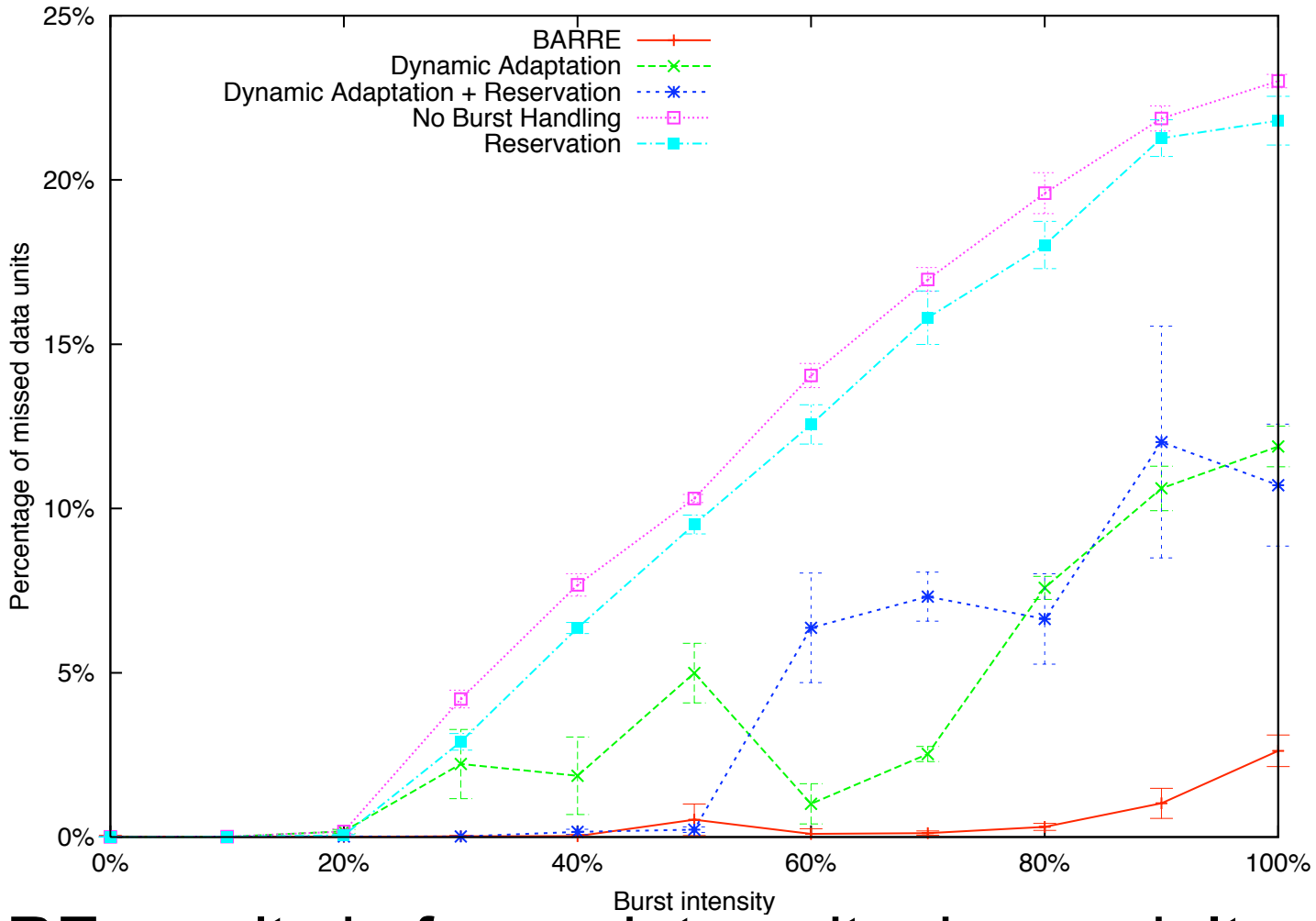
Combining multiple plans speeds up search, because of the small number of index points.

BARRE Operation



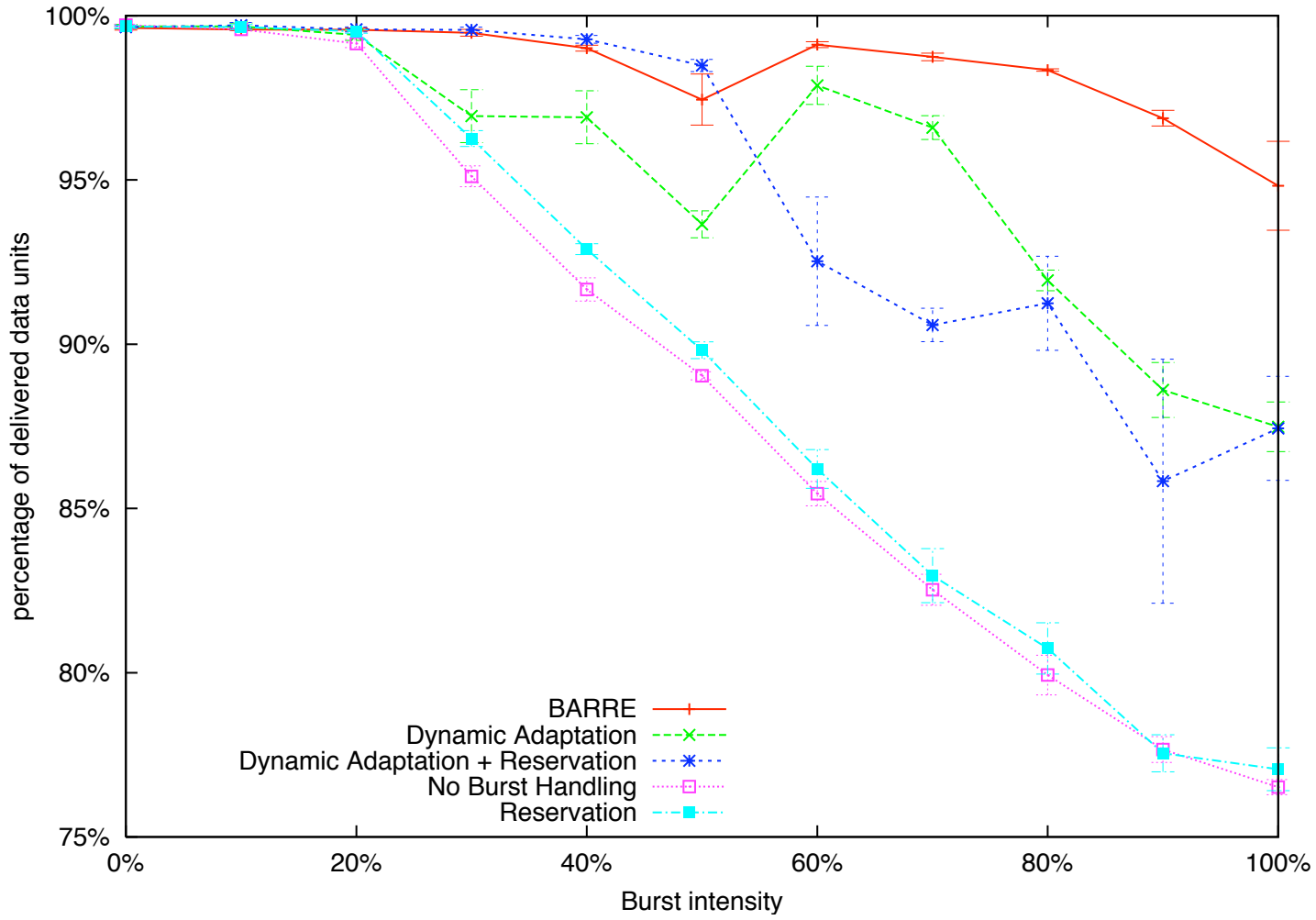
BARRE avoids missing data units during the burst and minimizes lost data units in the beginning of the burst. Also, resulting plan is “safer”, so it prevents future drops

Missed Data Units



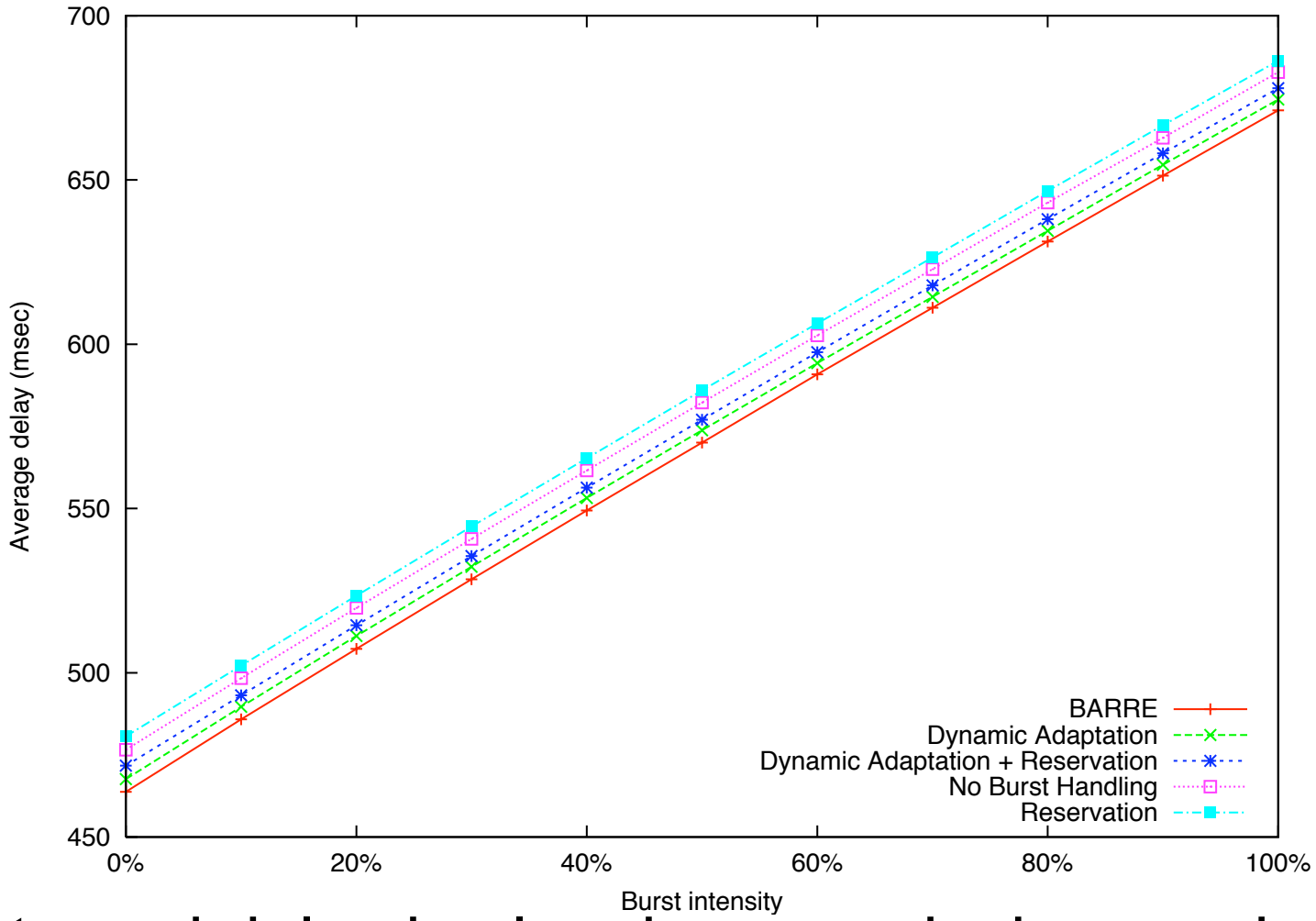
BARRE results in fewer data units dropped. It can sustain up to about 80% (vs. about 20%) application rate increase without dropping any data unit.

Data Units Delivered On Time



Despite splitting the work among many components, the arrival order of data units is not affected.

Average Delay



End-to-end delay is also decreased, since nodes are not overloaded. Other methods overload either powerful or underpowered nodes.

Conclusions - Future Work

- We proposed BARRE, a dynamic reservation scheme to address bursts in a dynamic stream processing system.
  - Reservation is based on application needs and readjusted according to system dynamics
- BARRE utilizes multiple nodes for each processing component, splitting the input rate among them whenever needed
- In our future work, we plan to evaluate BARRE in a fully dynamic environment with nodes entering / leaving the system.

Thank You!

Yannis Drougas, Vana Kalogeraki
Distributed Real-time Systems Lab
University of California, Riverside

{drougas,vana}@cs.ucr.edu
http://www.cs.ucr.edu/~{drougas,vana}