Task Allocation for Stream Processing with Recovery Latency Guarantee

Hongliang Li, Jie Wu, Zhen Jiang, Xiang Li, and Xiaohui Wei

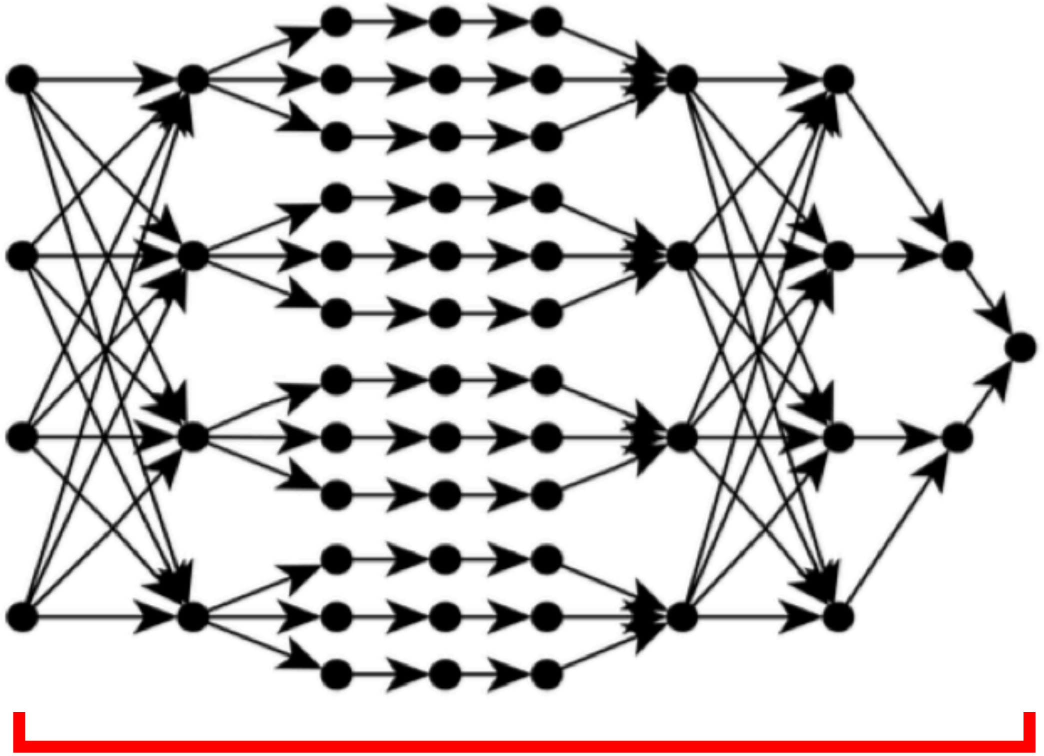
lihongliang@jlu.edu.cn

College of Computer Science and Technology, Jilin University, Changchun, China

Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA

Stream Processing Application and Model

- **Applications and Systems**
  - **Continuous, online, realtime or near realtime**
  - **High demand: data analyzing/monitoring for social network, production line, scientific experiment, etc.**
  - **Storm, Spark streaming, S4, Millwheel, Flink**
- **Stream Processing Model**
  - **On-the-fly, unable to obtain complete data beforehand**
  - **Stream topology**
    - **Workflow: tasks and links**
    - **Directed Acyclic Graph (DAG) of**
  - **Strict latency constraint: end-to-end**
- **Task allocation problem (failure-free)**
  - **Assign task/links to resource**
  - **Balance latency on each path, avoid bottlenecks**
  - **Optimization (bin packing, knapsack)**



Fault-tolerant for Stream Processing

- **Failure Effects**
  - **Vulnerable to failures: one-pass processing, in-memory processing, hard to recover from failures**
  - **Task failure: loss of internal state and data**
  - **Processor failure: multiple tasks on the processor fail at the same time**
- **Fault-tolerant Mechanisms**
  - **Active replication: high failure-free cost (Borealis)**
  - **Upstream backup + Checkpointing: recovery latency (Storm, Spark streaming, S4, Millwheel)**
- **Recovery latency**
  - **The time used to recovery from any failure, largest recovery latency of all tasks**
  - **Task allocation plan & stream topology**

Contributions

- **Failure Effect Model**
  - **Recovery latency, Topology and Allocation**
- **Task allocation problem considering recovery latency as a constraint**
- **Algorithms and results**

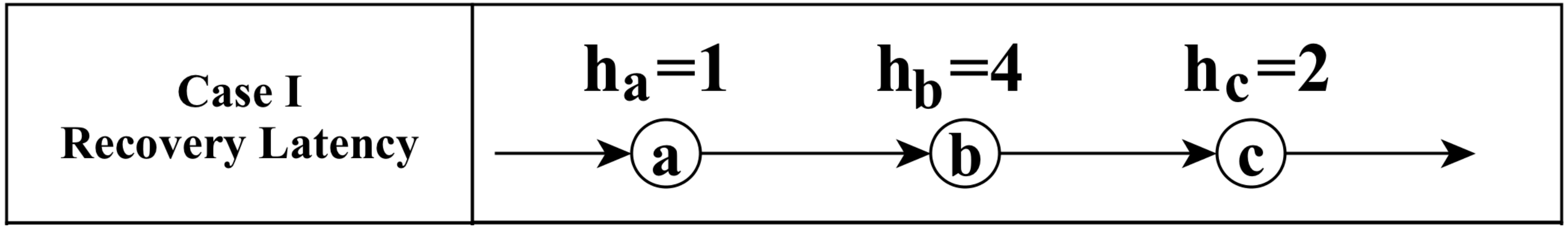
Failure Effect Model (1)

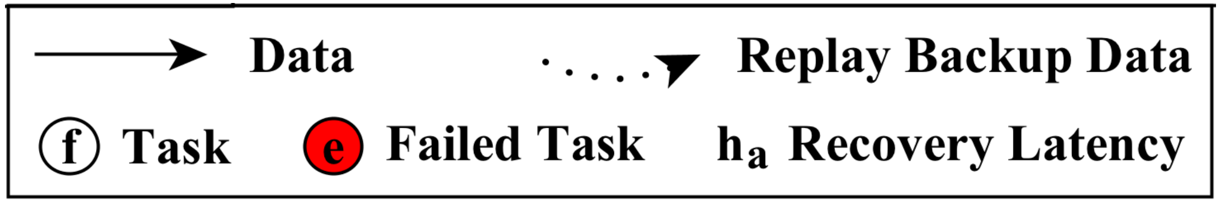
- **Recovery latency** $$h_v = r_v + t_v$$
  1) **Upstream latency (r): the time consumed on retrieving backup data**
     - **Can be cascading (related to topology and allocation)**
  2) **Reprocessing latency (t): the time spent on reprocessing data**
     - **Related to checkpoint interval (assumed to be given as an input)**

- **Cascading effect in correlated failures**
  - **Correlated failures: adjacent tasks failure together**
  - **Downstream task must wait for its dependent upstream task(s) to finish recovery**

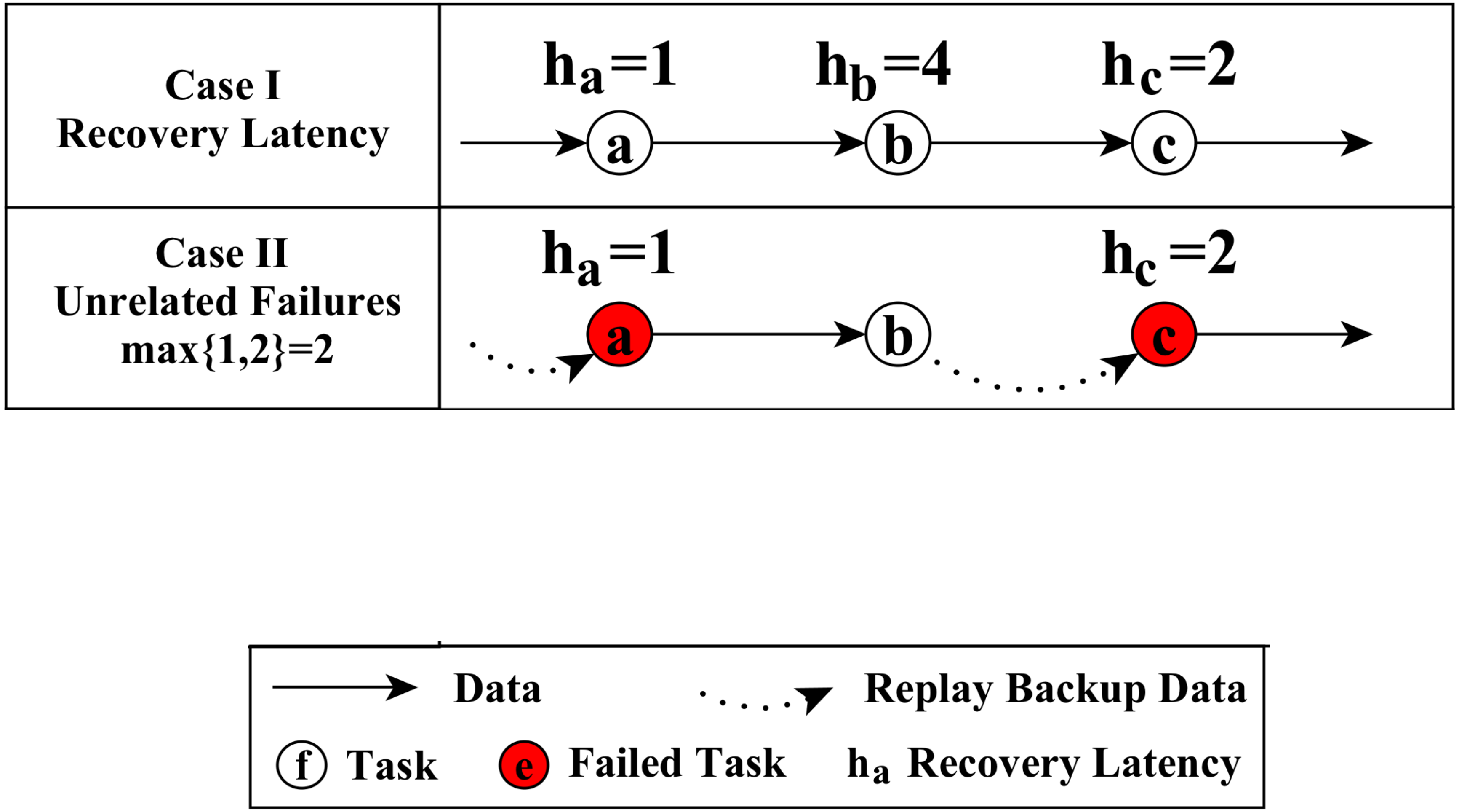
$$r_v := \begin{cases} 0 & \forall u \in U_v : f_u = 0 \\ \max\limits_{u \in U_v, f_u = 1} h_u & otherwise \end{cases}$$

Failure Effect Model (2)

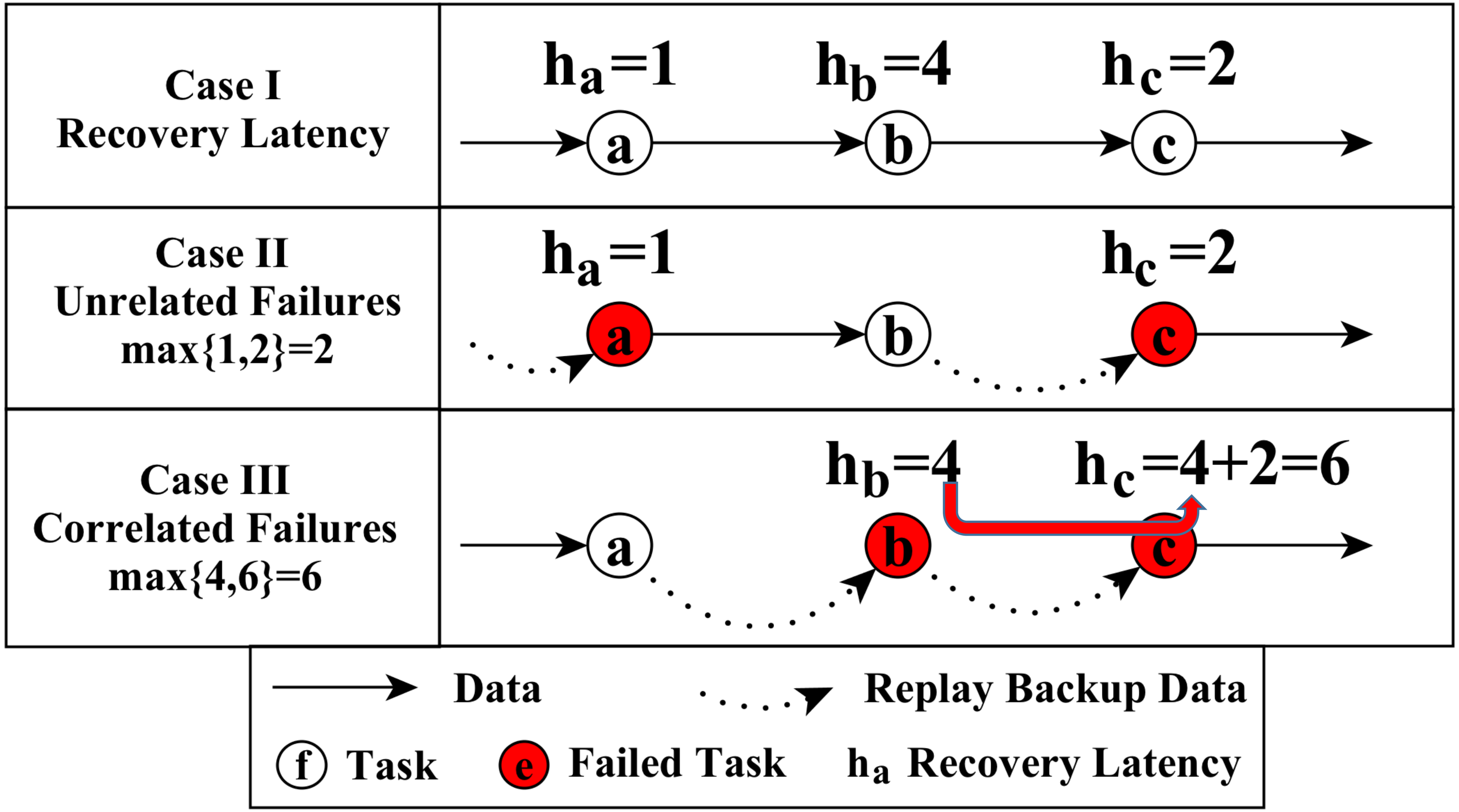




Failure Effect Model (2)



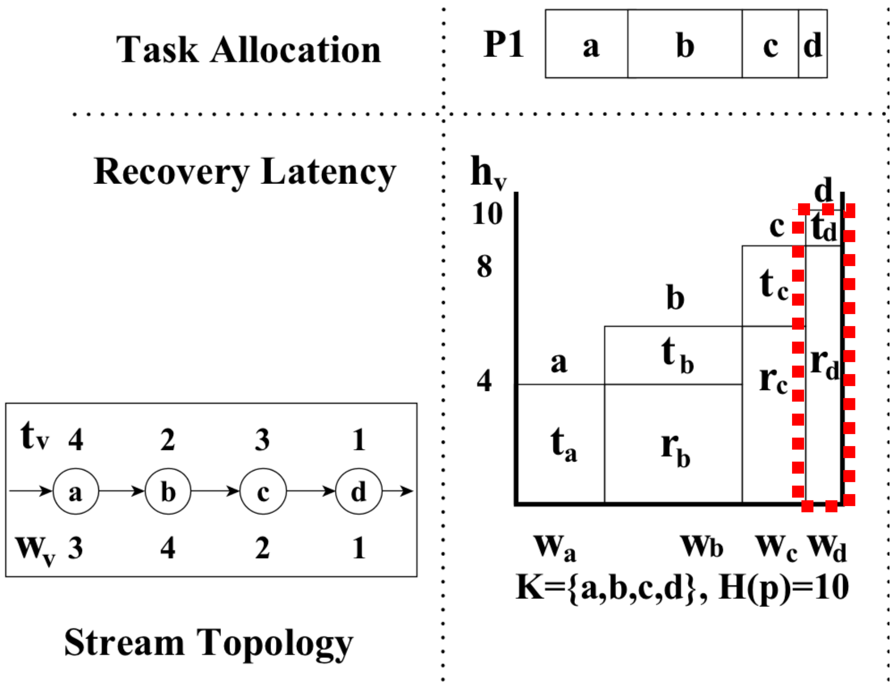
Failure Effect Model (2)



Task Allocation for Stream Topology (1)

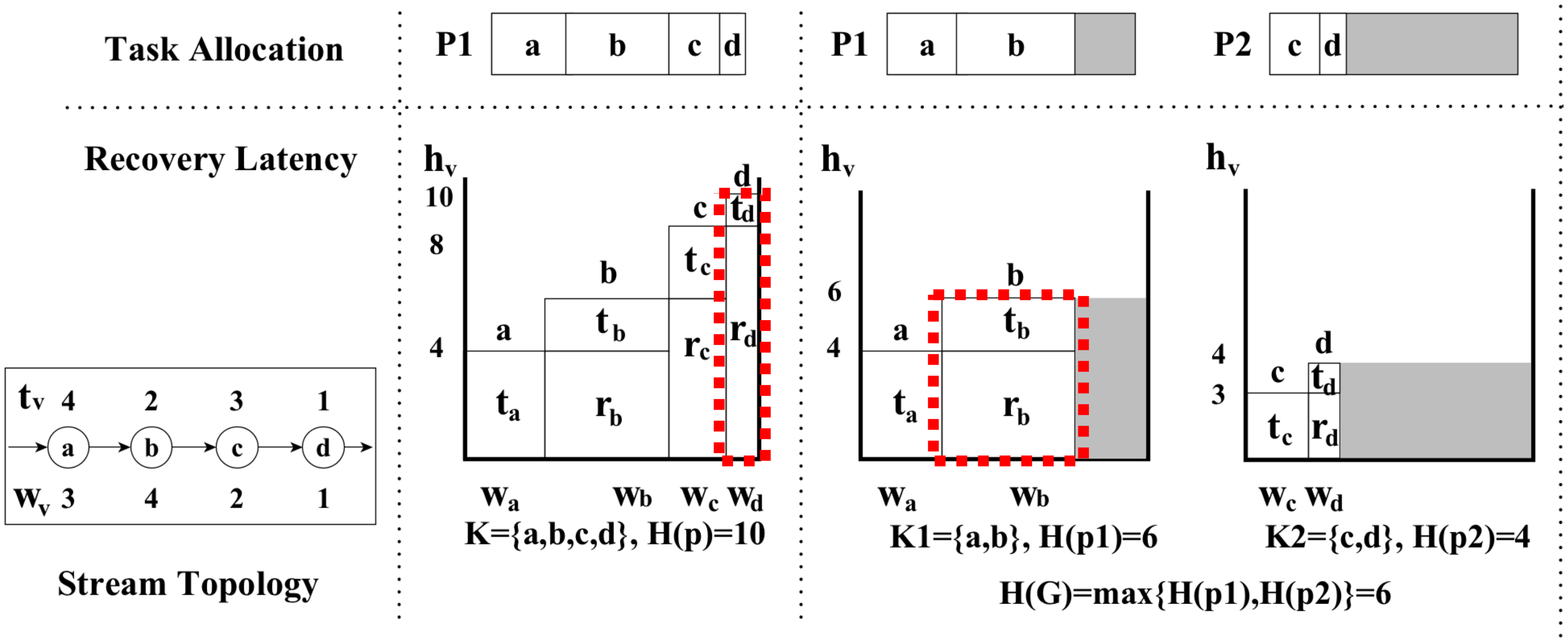
- **Correlated tasks: adjacent tasks on the same processor**
  - **Tasks on the same processor fail together in a processor-level failure**
- **Task allocations cause correlated tasks that affects the recovery latency in a processor-level failure.**
- **Assign tasks to processors (bin packing problem, NP-hard)**
  - **Task → Item, Processor → Bin**
  - **Height and Width**
    - **Item ---- the reprocessing latency and resource requirement of a task**
    - **Bin ---- the recovery latency constraint and the resource capacity of a processor**

Task Allocation for Stream Topology (2)



When K={a,b,c,d}, H(G)=10.
Case I

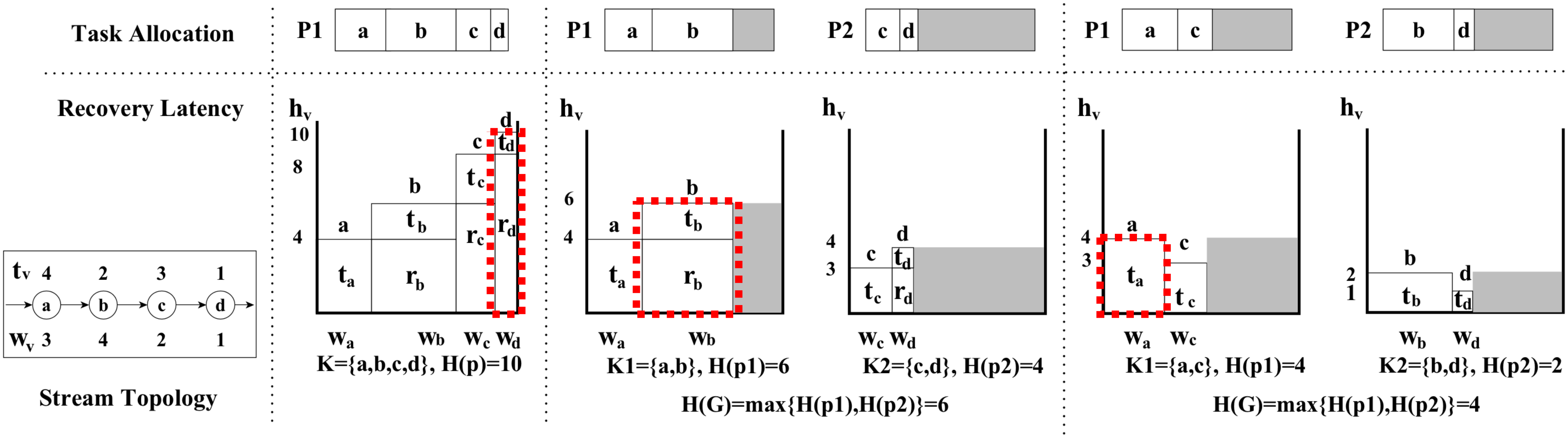
Task Allocation for Stream Topology (2)



When K={a,b,c,d}, H(G)=10.
Case I

When K1={a,b} and K2={c,d}, H(G)=6.
Case II

Task Allocation for Stream Topology (2)



When K={a,b,c,d}, H(G)=10.
Case I

When K1={a,b} and K2={c,d}, H(G)=6.
Case II

When K1={a,c} and K2={b,d}, H(G)=4.
Case III

How to allocate resource for tasks considering recovery latencies caused by correlated failures?

Task Allocation Problem with Recovery Latency Guarantee

$$\textit{minimize} \qquad Y = \sum_{j=1}^{m} y_j$$

$$\textit{subject to} \sum_{i=1}^{n} w_i x_{ij} \leq 1, \qquad j \in \{1, .., m\}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad i \in \{1, .., n\}$$

$$H(G) = \max_{v \in V} h_v \leqslant \bar{H}$$

$$y_j \in 0/1, \qquad \forall j \in \{1, .., m\}$$

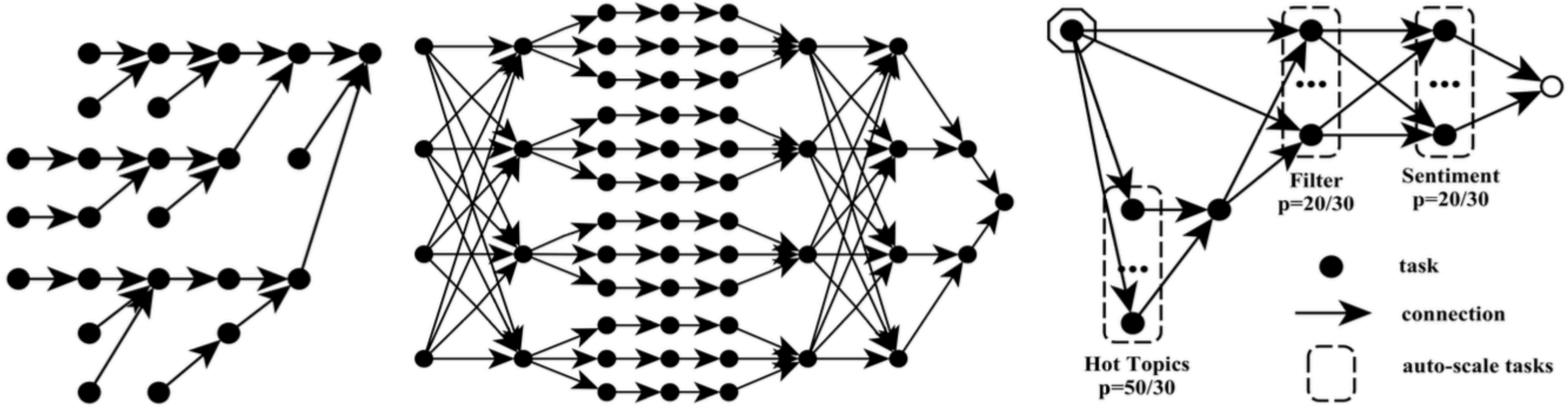
$$x_{ij} \in 0/1, \forall i \in \{1, .., n\}, \forall j \in \{1, .., m\}$$

- **Minimize amount of processors used**
- **Constraints**
  - **Resource capacity**
  - **Allocation**
  - **Recovery latency upper bound**

Algorithms

- **2SP-based greedy algorithms: NFDH, FFDN, BFDH**
  - **Sort items according to their heights (reprocessing latency)**
  - **Pack current item in the head of the queue to a bin according to NF/FF/BF strategy**
  - **Check bin height (recovery latency) constraint**
- **Observation: tasks with more adjacent tasks are more likely to cause correlated failures**
- **Proposed RTAP algorithm**
  - ***Sort* items in descending order according to packing “hardness”**
  - ***Partition* items into groups, avoid putting tasks that may break the recovery latency constraint in the same processor**
  - **Apply 1BP methods (NF/FF/BF) to each group**
- **Computational complexity** $O(n \cdot (\log(n))^2)$

Test Stream Topologies



(a) Tree topology (b) Sequential-dominated topology (c) Parallel-dominated topology

Type	$\|V\|$	$\|A\|$	α	β	γ
S-Tree	33	32	0.3	0.4	2
L-Tree	220	230	0.2	0.3	2
S-Guru	55	95	0.3	0.5	3.1
L-Guru	127	239	0.2	0.2	3.6
S-Senti	92	560	0.2	0.2	7
L-Senti	92	1050	0.2	0.2	22.5

- The *average width* of items, denoted by $\alpha = Ave(w_v)$.
- The *average height* of items, denoted by $\beta = Ave(t_v)$.
- The *average degree* of tasks, denoted by $\gamma = Ave(|D_v| + |U_v|)$.

Results

Algorithm	S-Tree		L-Tree		S-Guru		L-Guru		S-Senti		L-Senti	
	Y	ms	Y	ms	Y	ms	Y	ms	Y	ms	Y	ms
FFDN	30	22	125	130	32	22	82	14	38	22	45	23
NFDN	36	98	149	93	38	12	89	12	47	13	54	13
BFDN	29	11	113	131	32	17	81	14	38	18	45	16
RATP-FF	23	37	135	329	27	68	72	37	32	33	40	48
RATP-NF	44	51	165	268	39	40	84	49	43	38	56	47
RATP-BF	21	37	101	305	25	33	71	42	30	35	41	48

Note: **Y** is the amount of bins used.

- **ms-level execution times: applicable to task allocations and online adjustments**
- **15-25% less processors used comparing with 2SP-based benchmarks.**

Future Works

- **Extended version with more details in the approach and experiments**
- **Considering resource sharing among failed tasks and failure-free tasks**
- **Implementation and integration with stream processing systems**

Thank you very much!

- Hongliang Li
- lihongliang@jlu.edu.cn

- College of Computer Science and Technology, Jilin University, Changchun, China
- Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA