Integrated Recovery and Task Allocation for Stream Processing

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

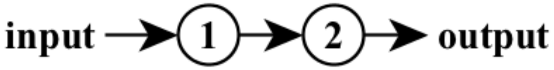
lihongliang@jlu.edu.cn

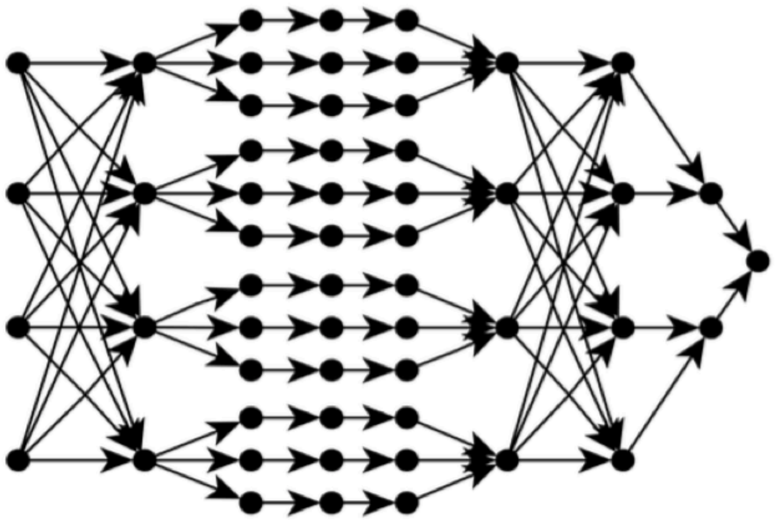
College of Computer Science and Technology, Jilin University, Changchun, China

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, in-memory computing**
  - **Stream topology**
    - **Workflow: tasks and links, Directed Acyclic Graph (DAG) of tasks**
  - **Strict throughput constraint: match the input rate to avoid data loss**





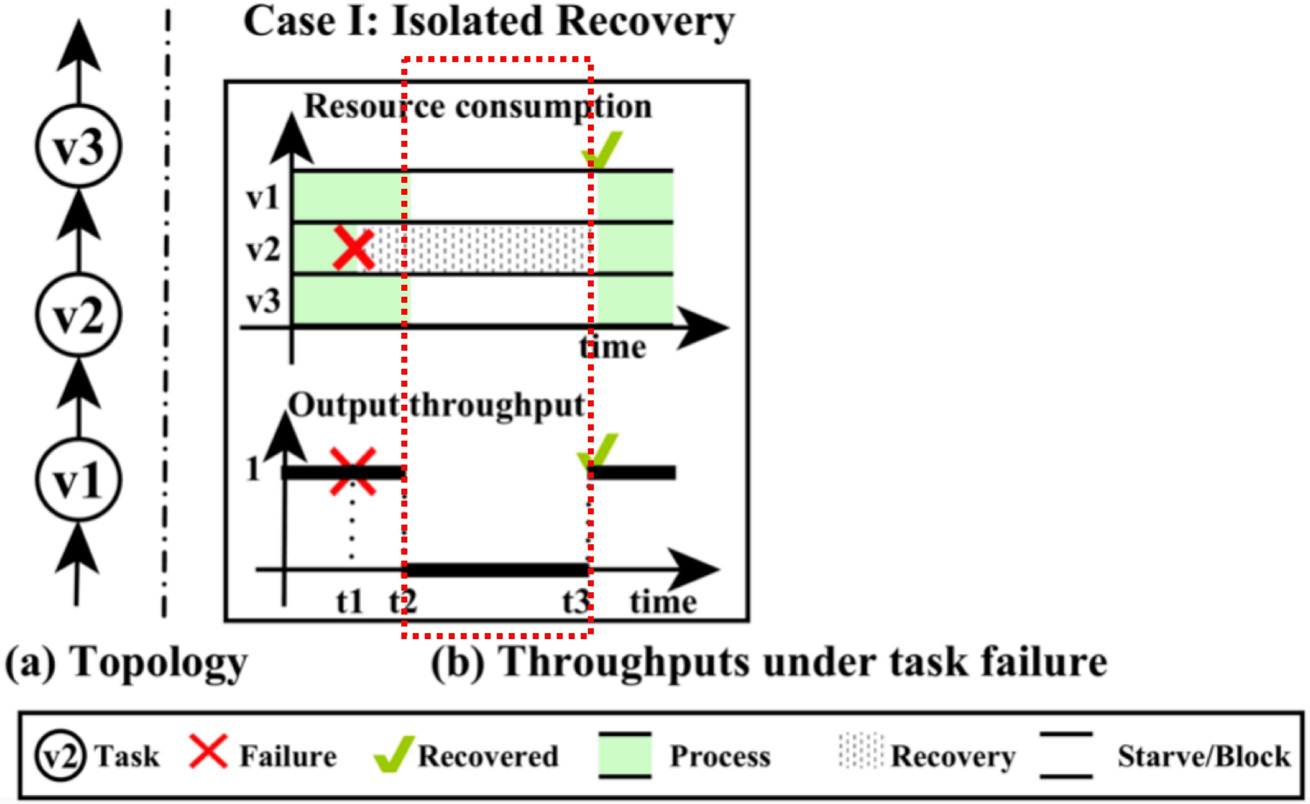
- **Task allocation problem (failure-free)**
  - **Assign task/links to resource (computing/network capacity)**
  - **Balance performance on each path, avoid bottlenecks**
  - **Optimization (bin packing, knapsack)**

Fault-tolerant for Stream Processing

- **Vulnerable to failures**
  - **One-pass processing, in-memory processing, hard to recover from failures**
  - **Task failure: loss of internal state and data**
- **Fault-tolerant Mechanisms**
  - **Active replication: high failure-free cost (Borealis)**
  - **Upstream backup + Checkpointing: recovery latency (Storm, Spark streaming, S4, Millwheel)**
- **Failure Effect**
  - **Cost: reprocess backup data from upstream**
  - **Suspend from producing new data, affects throughput or even cause an application-level halt**

Different recovery schemes

- **Isolated recovery model**
  - **Exclusive resources**
  - **Failure-free tasks can be starved/blocked**
  - **Failure can cause an application-level halt**

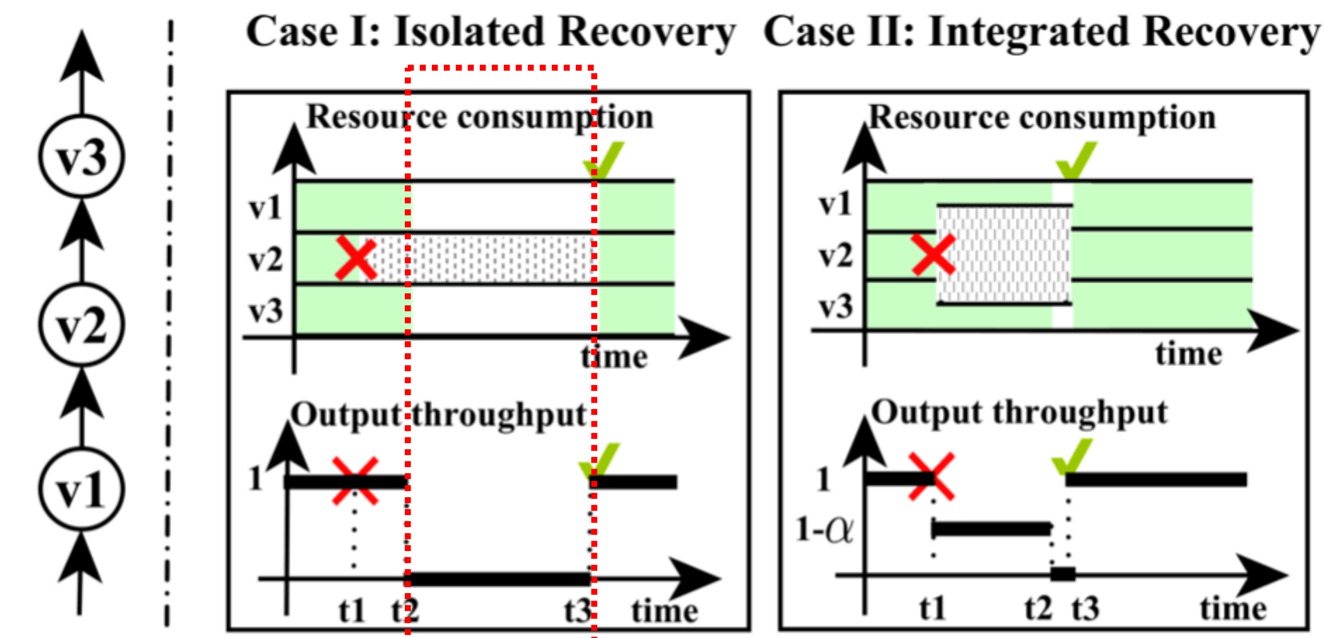


(a) Topology (b) Throughputs under task failure

Note: α and t_1-t_3 represent the slowdown and delay of a recovery, respectively. v_1 and v_3 process buffered data in t_1-t_2. The processing is halted in t_2-t_3.

Different recovery schemes

- **Isolated recovery model**
  - Exclusive resources
  - Failure-free tasks can be starved/blocked
  - Failure can cause an application-level halt
- **Integrated Recovery Model (IRM)**
  - Share resource from failure-free tasks
  - Accelerate the recovery
  - Reduce performance degradation
  - Can even avoid starvation/blocking and performance degradation (buffer setting)



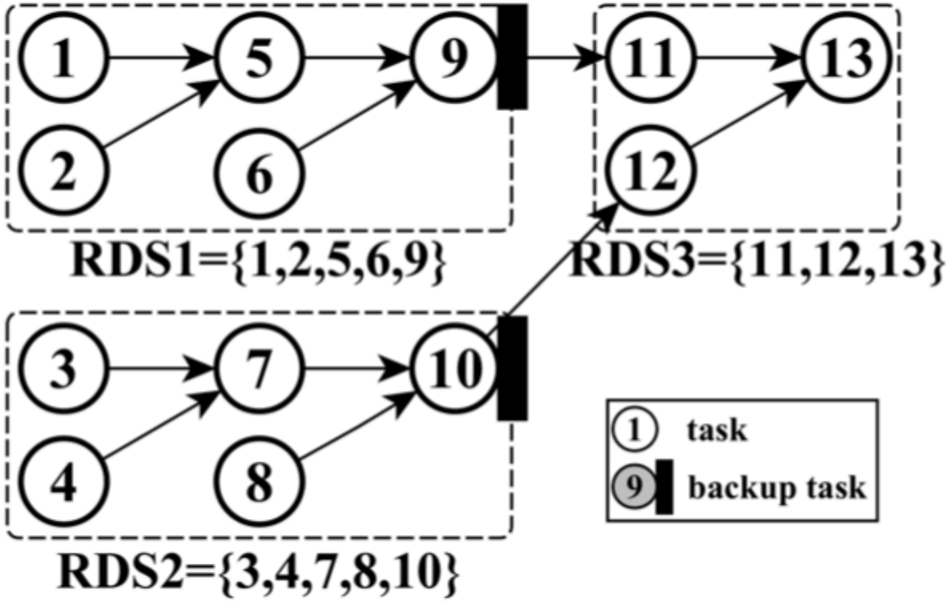
(a) Topology (b) Throughputs under task failure

Note: α and t_1-t_3 represent the slowdown and delay of a recovery, respectively. v_1 and v_3 process buffered data in t_1-t_2. The processing is halted in t_2-t_3.

Contributions

- **Novel Integrated Recovery Model (IRM)**
  - **Enable resource sharing between failed and failure-free tasks**
  - **Support fast and seamless recoveries**
- **Cost-aware Task Allocation Problem (CTAP)**
  - **Consider recovery cost as part of resource requirement, besides failure-free processing cost, during task allocation**
  - **Guaranteed processing performance during recovery (slowdown ratio)**
- **Algorithms and results**

Integrated Recovery Model



- **Upstream Backup Model**
  - **FT Configuration: set up backup tasks**
  - **Upstream replay and recovery**
- **Recovery Dependent Set (RDS)**
  - **A subset of task in a stream topology, divided by backup tasks**
- **Upstream Recovery Dependent Set (URDS)**
  - **Task *v*'s upstream tasks that are in the same RDS**
- **Recovery cost**
  - **Task isolated recovery cost** δ_v **: related to checkpointing interval**
  - **Task *v* on processor *c*** $\Delta_{v/c} := \sum_{u \in URDS_v, \Phi(u)=c} \delta_u$
  - **On processor *c*** $\Delta_c = \max_{v \in V} \{\Delta_{v/c}\}$

Cost-aware Task Allocation Problem (1)

- **Failure-free Task Allocation Problem**

  The failure-free task allocation problem seeks a TAP, denoted by $\Phi: V \rightarrow C$, that assigns a set of n tasks (V) to a set of p identical processors ($C = \{c_i | i \in 1, \ldots, p\}$).

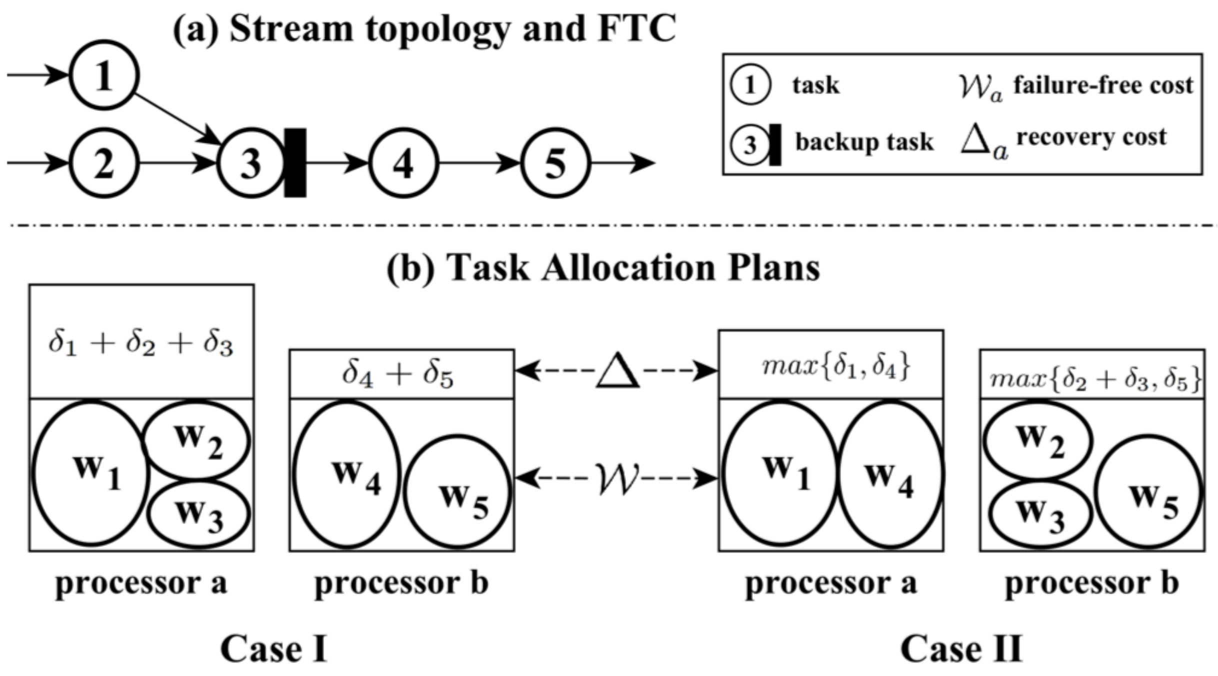
- **Failure-free cost (processing)**

$$\mathcal{W}_c := \sum_{\Phi(v)=c} w_v, c \in C$$

- **Recovery Cost** $\Delta_c = \max_{v \in V}\{\Delta_{v/c}\}$

- **Slowdown Ratio**

$$\alpha_c := \begin{cases} 0 & 1 - \Delta_c \geq \mathcal{W}_c \\ 1 - \frac{1-\Delta_c}{\mathcal{W}_c} & otherwise \end{cases}, c \in C$$



Cost-aware Task Allocation Problem (2)

- **Modeling (Packing problem)**
  - **Target: minimize the used processors**
  - **Constraints:**
    - **Capacity** $\mathcal{W}_j = \sum_{i=1}^{n} w_i z_{ij} \leq 1,\ j \in \{1, .., p\}$
    - **Performance** $\max_{c \in C}\{\alpha_c\} \leqslant \bar{\alpha}$

$$minimium \quad \sum_{j=1}^{p} x_j$$

$$subject\ to \quad \sum_{j=1}^{p} z_{ij} = 1, \qquad i \in \{1, .., n\}$$

$$\mathcal{W}_j = \sum_{i=1}^{n} w_i z_{ij} \leq 1,\ j \in \{1, .., p\}$$

$$\max_{c \in C}\{\alpha_c\} \leqslant \bar{\alpha}$$

$$x_j = 0/1, \forall j \in \{1, .., p\}$$

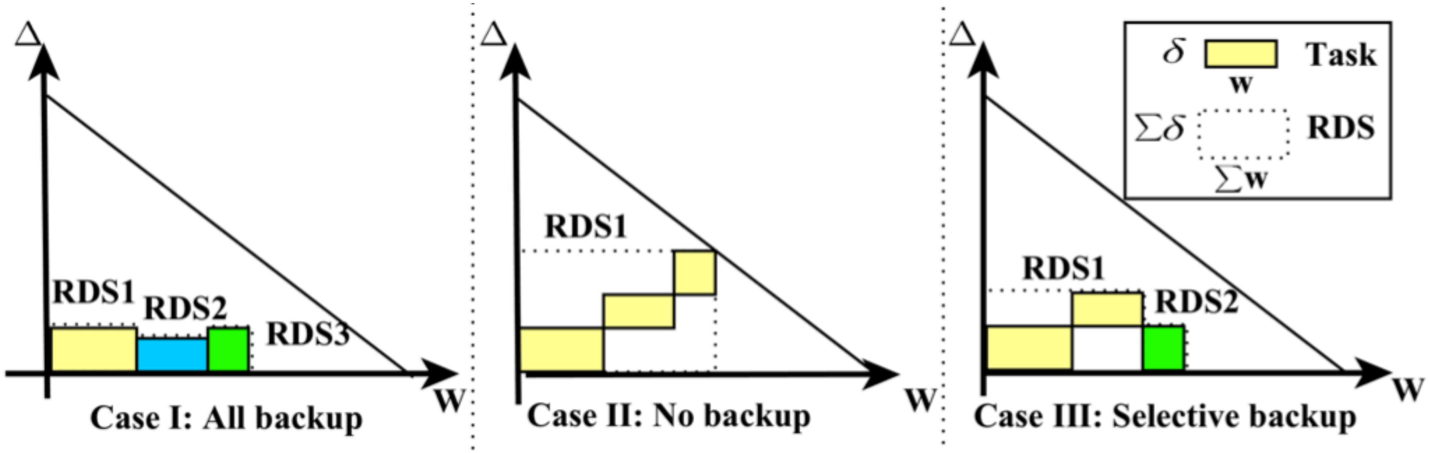
$$z_{ij} = 0/1, \forall i \in \{1, .., n\}, \forall j \in \{1, .., p\}$$

Cost-aware Task Allocation Problem (3)

- **Discussion**
  - **When upper bound of slowdown ratio $\bar{\alpha}$ is given, Δ_c is inversely linear proportional to $\mathcal{W}_c$**

    $$\Delta_c = 1 - (1 - \bar{\alpha}) \cdot \mathcal{W}_c$$

  - **(a) All tasks are backup tasks, one task in each RDS, Bin Packing Problem (BPP)**
  - **(b) No backup tasks, all tasks are in one RDS, 2D vector packing**



Algorithms

- **BPP-based greedy algorithm as benchmark (BestFit),** $O(n \log n)$
  - **Sort items in descending order according to their failure-free cost** w_v
  - **Pack item in the head of the queue to a bin according to BF strategy**
  - **Check capacity and slowdown ration constraints**
- **Observation**
  - **When tasks in the same RDS packed into one processor, large recovery cost can be introduced; recovery cost accumulated only among tasks in the same RDS**
- **Proposed Heuristic algorithm**
  - ***Partition tasks according to RDSs;***
  - ***Sort RDSs according to their sizes in ascending order;***
  - ***Choose an item from an RDS and assign the task to a processor that causes the smallest potential recovery cost.***
- **Computational complexity** $O(n \cdot (\log(n))^2)$

Test Settings

- **Stream Topologies**

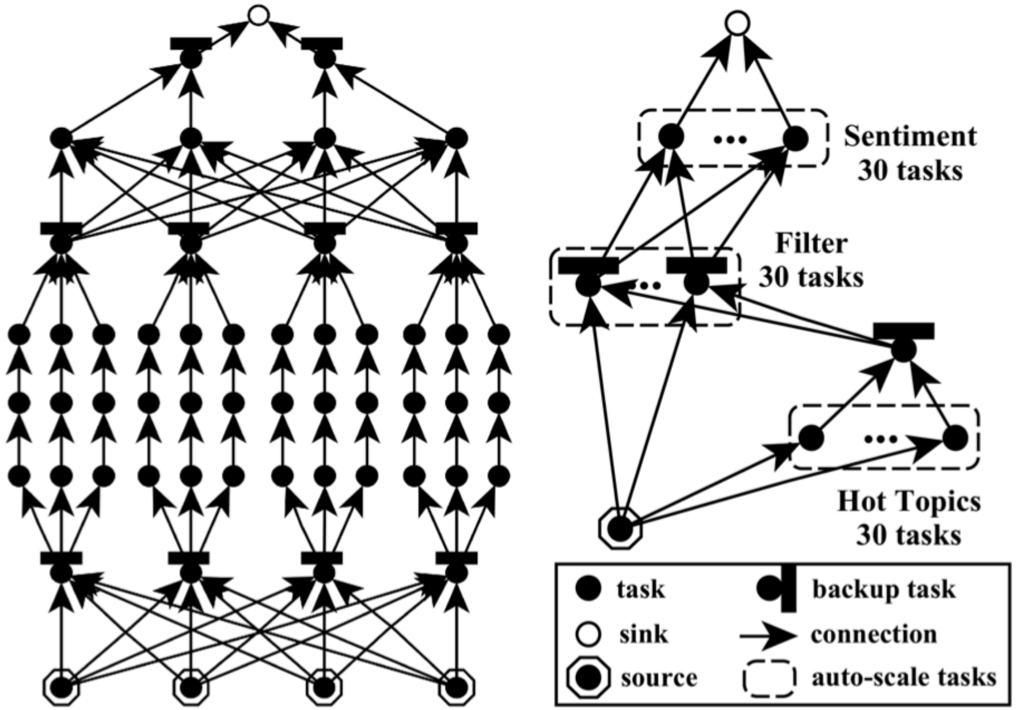
Type	Description
In-Tree	One adjacent downstream task. $\|V\| = 400$, $\|A\| = 400$.
Sequential-dominated	DAG with long paths [34]. $\|V\| = 55$, $\|A\| = 95$.
Parallel-dominated	Auto-scale tasks [11]. $\|V\| = 93$, $\|A\| = 1050$.

- **Backup Settings**

Type	Description
A	All task are backup tasks [14], [15].
B	Only the input streams have backups [5], [16].
C	Selected tasks are back-up [19].

- **Comparing Approaches**

Algorithm	Description
Failure-free	BF packing algorithm that does not consider task failures.
Greedy	Algorithm based on BPP strategy (BF)
Heuristic	Heuristic based on RDSs and current recovery cost Δ_c



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

Results

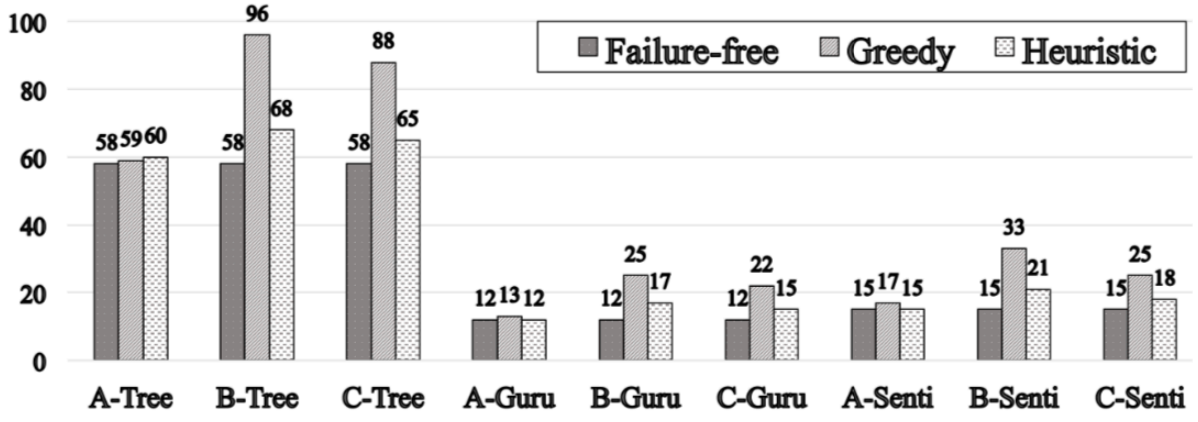


Figure 6: Number of processors.

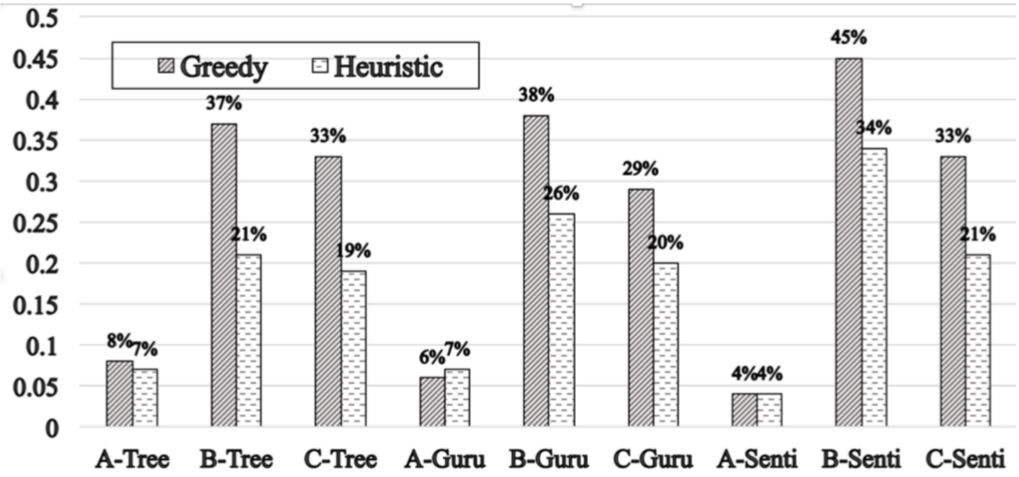


Figure 7: Average recovery costs (%).

- **Extra Processors**: 48% (greedy) and 14% (heuristic) extra processors are used on average comparing with failure-free task allocations.
- **Resource Utilization ($\bar{\alpha}$ = 30%)**: The average recovery costs in the greedy and heuristic approaches are 26% and 17%, respectively. The resource utilization ratio are 74% and 83% respectively.

Results

Table V: Execution Time (ms) of Different Approaches.

Algorithm	A-Tree	B-Tree	C-Tree	A-Guru	B-Guru	C-Guru	A-Senti	B-Senti	C-Senti
Failure-free	89	88	88	12	13	12	37	42	38
Greedy	93	128	91	25	36	30	49	59	55
Heuristic	170	138	148	50	46	55	118	108	99

- **ms-level execution times**
- **Applicable scenario**
  - **generate efficient task allocation decisions with guaranteed recovery performance, i.e. an upper bound of throughput slowdown**
  - **ensure continuous results without a suspension during any task-level failure recovery (with proper buffer settings)**
  - **provide quick feedbacks for FT configuration solutions**
  - **serve as a tool to analyze the performance of a system**

Conclusions

- **Integrated Recovery Model (IRM) that allows resource sharing between a recovering task and the failure-free tasks on a processor. It enables fast and seamless recoveries.**

- **We introduce a novel task allocation problem under IRM.**

- **Algorithms and experimentations**

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