Task Allocation for Stream Processing with Recovery Latency Guarantee

Hongliang Li*†, Jie Wu†, Zhen Jiang‡, Xiang Li*, and Xiaohui Wei*
*College of Computer Science and Technology, Jilin University, Changchun, China
†Department of Computer and Information Sciences, Temple University, Philadelphia, USA
‡Department of Computer Science, West Chester University of Pennsylvania, West Chester, USA
Email: lihongliang@jlu.edu.cn, jiewu@temple.edu, zjiang@wcupa.edu, {lxiang, weixh}@jlu.edu.cn

***Abstract*—Stream processing applications continuously process large amounts of online streaming data in real-time or near real-time. They have strict latency constraints, but they are also vulnerable to failures. Failure recoveries may slow down the entire processing pipeline and break latency constraints. Upstream backup is one of the most widely applied fault-tolerant schemes for stream processing systems. It introduces complex backup dependencies to tasks, and increases the difficulty of controlling recovery latencies. Moreover, when dependent tasks are located on the same processor, they fail at the same time in processor-level failures, bringing extra recovery latencies that increase the impacts of failures. This paper presents a correlated failure effect model to describe the recovery latency of a *stream topology* in processor-level failures for an allocation plan. We introduce a Recovery-latency-aware Task Allocation Problem (RTAP) that seeks task allocation plans for stream topologies that will achieve guaranteed recovery latencies. We present a heuristic algorithm with a computational complexity of $O(n \log^2 n)$ to solve the problem. Extensive experiments were conducted to verify the correctness and effectiveness of our approach.**

***Keywords*-Stream processing, task Allocation, fault-tolerance, upstream backup, recovery latency**

I. Introduction

Stream processing applications are in high demand in various areas, including analyzing social networks, trading high-frequency stocks, and monitoring and controlling production lines. This has led to a rapid increase in the popularity of a new computing paradigm known as the Stream Processing Model (SPM) [1], [2].

Stream processing applications share a common characteristic – strict latency constraint. That is, they must provide a fast and accurate response. A stream processing application takes data stream(s) as input, performs a series of predefined functions (*tasks*), and generates output in the form of data streams again. It is usually modeled as a Directed Acyclic Graph (DAG) of *tasks*, i.e. *Stream Topology* [3]. The processing latency of a stream topology is the end-to-end elapsed time from the entrance of a set of input data to the emission of the corresponding output data [3]. Existing works focus on balancing latencies among multiple paths by assigning each task appropriate physical resources; this is know as the *task allocation* problem [3]–[5] for SPM.

However, when extra time is needed for the recovery of a failed task, especially when a hardware issue forces the recovery of all the tasks allocated to a processor, the slowdown of a single task and the possible accumulated impact of all delayed tasks will suspend the processing. This will slow down the entire processing and may break the latency constraints.

Stream processing scheme performs "one-pass" processing over stream data on the fly without storing them [6], which is entirely different from the traditional "process-after-store" mode. The SPM can be more vulnerable to failures [7], [8] than other big-data processing schemes (e.g. [9]). This unique characteristic poses a novel fault-tolerant problem [1], [10].

In recent years, the upstream backup scheme [10]–[13] has been widely applied because of its small Fault-Tolerant (FT) overhead (compared to the one needed in the active replication scheme [14]). Each task can maintain a backup of its output data for its downstream tasks. Upon failure, all of the upstream backup tasks replay backup data, and the recovering task reprocesses the data to recover the previous status; this introduces task *recovery latency* [10], [11]. The recovery latency of a stream topology is the largest recovery latency of its tasks [8], [14].

However, the impact of correlated failures is not considered in literature. An allocation plan may put multiple tasks on one processor to share physical resource [3], [4]. This is very common in practice due to the use of the modern CPU core. In such a case, tasks located on the same processor fail together during a processor-level failure. The recovery latency of a stream topology is affected by the task allocation plan.

The purpose of this paper is to study the relationship between recovery latency and task allocation plan and to present a comprehensive approach to compute task allocation plans that provides recovery latency guarantee. The main contributions are summarized as follows:

- We present a quantitative model that describes the relationship between the recovery latency and the task allocation plans of a stream topology.
- We introduce the Recovery-latency-aware Task Allocation Problem (RTAP) and discuss how it differs from classic task allocation problems.
- We propose a heuristic algorithm based on the topology information to compute task allocation plans with recovery latency guarantee, and present simulation results to verify the correctness and effectiveness of our approach.

2168-9253/17 $31.00 © 2017 IEEE
DOI 10.1109/CLUSTER.2017.10

II. Problem Formulation

A. Task Allocation Problem for Stream Topology

A stream topology is usually modeled as a DAG $G(V, A)$ [3], where the vertices $V = \{v_i | i \in 1, .., n\}$ represent *tasks* and arcs $A = \{a(v_i, v_j) | v_i, v_j \in V\}$ represent connections between tasks. For each task $v \in V$, resource requirement (w_v) is given, reflexing the computing capacity needed by the task according to its input data rate [3]. The task allocation problem in failure-free scenario seeks to map all tasks to processors to a minimum set of processors, while satisfying all the resource requirements of the task.

B. Correlated Failure Effect Model

We assume that each task performs *upstream backup* [10], [11]. Upon failure, a task will restart and reset to its previous backup state. Corresponding backup data are then replayed from the upstream task(s) to the recovering task.

The recovery process of a task introduces a recovery latency ($h_v = r_v + t_v$) that consists of two parts [8]: (1) *upstream latency* (r_v), the time consumed retrieving backup data, and (2) *reprocessing latency* (t_v), the time spent reprocessing data. The former is related to the task's checkpoint interval, and it can be estimated using the methods in [8], [11], [15]. In this paper, we assume that t_v, $v \in V$, is given as an input.

We propose the correlated failure effect model for reprocessing latency. Assuming that task v is recovering, and U_v is the set of adjacent upstream tasks of v, task v can obtain backup data right away ($r_v = 0$). Otherwise, when upstream task $u \in U_v$ fails at the same time as v, the backup data on u is lost. Face with such correlated failures, the downstream task (v) must wait for its dependent upstream task ($u \in U_v$) to finish recovery before necessary backup data becomes available again. Let the binary variable f_v denote the healthy status of a task. We define the upstream latency recursively as Eq.(1). When multiple adjacent tasks on the same path fail together, the effect of the delay can be cascading.

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

C. Recovery Latency under Processor Failure

K_p denotes the set of tasks placed on processor p, $K_p = \{v | v \in V, \Phi(v) = p, p \in P\}$. When tasks in K_p fail together, $r_v = \max\{\max\limits_{u \in U_v, \Phi(u) = \Phi(v)} h_u, 0\}$. The recovery latency of a processor p under such a failure is $H(p) = \max\limits_{v \in K_p} h_v$. The *reprocessing latency* of a stream topology is equal to the largest recovery latency of any processor where $H(G) = \max\limits_{p \in P} H(p) = \max\limits_{v \in V} h_v$. Note that the above model supports scenario where the processors have independent failure rates and concurrent fails.

D. Task Allocation with Recovery Latency Guarantees

Problem RTAP (*Recovery-latency-aware Task Allocaiton Problem (RTAP)*). *Given a stream topology graph $G(V, A)$, the available processor set $P = \{p_i | i \in 1, .., m\}$, and the recovery latency upper bound $\bar{H}$, find the task allocation $\Phi = V \rightarrow P$ according to task weight w_v that occupies the minimum number of processors while satisfying recovery latency constraints.*

The objective function seeks to minimize the number of occupied processors, subject to the resource and recovery latency constraints. Such a RTAP problem is NP-hard since it generates the classic Bin Packing Problem (BPP) [16]. From this point on, we use the terms items and tasks interchangeably, and bins and processors are also used interchangeably. A processor and its resource capacity are hereafter referred to as a bin and its width, and a task and its weight are hereafter referred to as an item and its width. The height of a bin represents the recovery latency threshold $\bar{H}$. The height of an item corresponds to the reprocessing latency (t_v) of a task.

As shown in Fig. 1, different task allocation plans may lead to different recovery latencies. Case I packs all tasks into one processor, i.e. $K = \{a, b, c, d\}$, like typical 2SP approaches. It does not waste the width of a bin, but causes correlated task failures that introduce high upstream latencies. Cases II and III spread out tasks to avoid such a situation, but they introduce resource waste. Case III achieves the smallest recovery latency. There is a trade-off between the recovery latency of the stream topology and the amount of resources in use.

We assume that both the height and width of a bin (item) are normalized to 1. Variable x_{ij} represents the allocation decision for mapping task i to processor j. Variable y_j represents whether processor j is used. A valid model for the RTAP corresponds to Eqs.(2)-(7).

$$minimize \qquad Y = \sum_{j=1}^{m} y_j \tag{2}$$

$$subject\ to \sum_{i=1}^{n} w_i x_{ij} \leq 1, \qquad j \in \{1, .., m\} \tag{3}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad i \in \{1, .., n\} \tag{4}$$

$$H(G) = \max_{v \in V} h_v \leqslant \bar{H} \tag{5}$$

$$y_j \in 0/1, \qquad \forall j \in \{1, .., m\} \tag{6}$$

$$x_{ij} \in 0/1, \forall i \in \{1, .., n\}, \forall j \in \{1, .., m\} \tag{7}$$

III. Approach

This RTAP is a new problem. The most similar variation of the BPP to the RTAP is the Two-Dimensional Strip Packing Problem (2SP) [17]. The main challenge in RTAP is that item height (recovery latency) is related to both the partial solution (task allocation plan) and the stream topology. There are few efficient exact algorithms for the BPP. Any exhaust search algorithm for the RTAP will be impractically time-consuming. We first propose three greedy algorithms based on well-known 2SP approaches; these are used as benchmarks in experiments. We then propose a fast topology-aware heuristic algorithm.

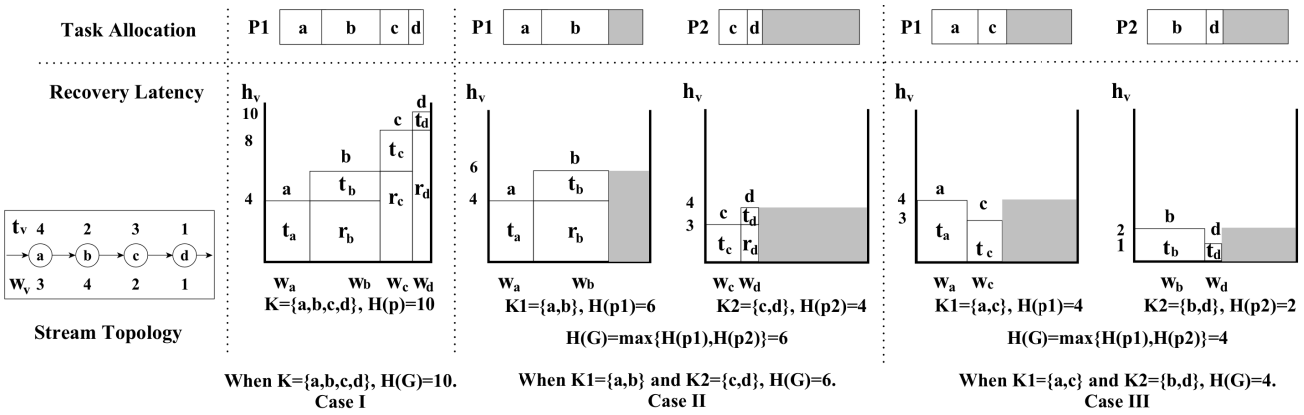


Figure 1: Trade-off between latency and resource: an examples of different task allocation plans.

A. Extension of 2SP Approaches [17] considering Recovery

Next-Fit Decreasing Height (NFDH), *First-Fit Decreasing Height* (FFDH), and *Best-Fit Decreasing Height* (BFDH) are widely-applied "level-oriented" algorithms [17] for the 2SP problem. We extend these 2SP algorithms and design greedy algorithms NFDH, FFDH, and BFDH. Items are sorted in descending order according to their heights, breaking ties by decreasing the width. These algorithms then pack one item at each step based on the 2SP strategies used. At each round, before putting an item into a bin, a function is applied to update the current level height $H(p)$ – the height of the highest item in the bin. This function introduces an extra $O(\log n)$ time complexity. Then, the estimated recovery latency of the current item is examined according to Eq.(5). The time complexity of all these algorithms is $O(n \log^2 n)$.

B. Comprehensive Solution with a Heuristic Algorithms

We propose a heuristic A that sorts items in descending order according to packing "hardness". We design our algorithm according to the observation that tasks with more adjacent tasks are more likely to cause correlated failures. We introduce a new metric for sorting the items: the Weighted Upstream Degree (WUD). $WUD_v = \frac{|U_v|}{n} \cdot \sum_{u \in U_v} t_u$. When a task is packed into a bin with its adjacent upstream tasks, its upstream latency increases. This may make the task unfit for the bin. Even worse, when a task is packed with its adjacent downstream tasks, extra heights are introduced to the downstream tasks. This may result in a change in the current partial solution and in the backtracking of tasks as they become unfit for the bin.

In order to avoid these backtracking situations and accelerate the packing process, items are first ordered by their WUD values, breaking ties by breadth-first traversal orders, and then partition into groups. The partitioning seeks to avoid putting tasks that may break recovery latency in the same group. Then the algorithm apply One-dimensional Bin Packing (1BP) strategies [17]. We implement three heuristic algorithms (A-NF, A-FF, and A-BF) using different 1BP strategies to explore the effects of different packing methods.

Algorithm 1 $A\ (G, P, \bar{H})$

Input: A stream topology graph $G(V, E)$, processor set P, and recovery latency constraint $\bar{H}$
Output: Task allocation Φ

```
Sort items according to combined weight degree (WUD);
Partition items into Groups {G'1, G'2, .., G'q};
for i ∈ {1, .., q}, Group G'i; do
    Apply 1BP on G'i according to constraint (5) to get Φi;
    Φ := Φ ∪ Φi ;
end for
return Φ;
```

Computing the weighted upstream degree for each item and sorting items introduces $O(\log n)$ computations. The same thing happens when packing each item that current packing solution is examined and items' estimated heights are updated, costing $O(\log n)$ extra computations than 1BP approaches. It is easy to prove that the computational complexity of $Algorithm^{RTAP}$ is $O(n \log^2 n)$, but due to space limitations, we omit the details in this paper.

IV. Simulation Results

A. Experimental Settings

We conduct simulations to illustrate: (1) the performance of the proposed algorithms compared to 2SP-based algorithms, (2) the efficiency of our approach for different types of stream topologies. We use three 2SP greedy algorithms (NFDH, FFDH, and BFDH) as benchmarks, and test the proposed three heuristic algorithms (A-NF, A-FF, and A-BF).

Three types of stream topologies from stream processing applications are used as our previous paper [8]. "Tree" topology is a topology where each task has exactly one downstream task. "Guru" topology (SignalGuru) is a sequential-dominated topology that has longer paths than others, and "Senti" topology (Twitter Sentiment) is a parallel-dominated topology with a large amount of parallel tasks. Each of the three topologies is further extended into two test cases with different amounts of tasks and edges, denoted as S-Type and L-Type.

B. Experimental Results

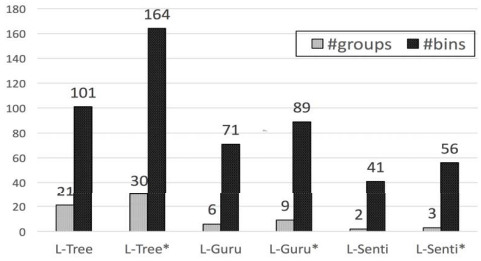
We test the proposed algorithms on all test instances to measure performance, in terms of the number of processors (#bins) used and the algorithm execution time (milliseconds), as shown in Table I. Among the three greedy 2SP-based algorithms, A-NF performs the worst while A-BF has the best result. Using a group function based on topology information, the heuristic algorithms proposed in this paper outperform the 2SP-based algorithms. A-BF uses the best-fit strategy after a grouping process and has the best results. Overall, the proposed group-based heuristics use 15-25% fewer processors than the 2SP-based benchmarks. The grouping function involves an extra searching procedure in each task, and this can be time-consuming when the degree of each task is large. However, the proposed algorithms have a polynomial time complexity so that they can compute task allocation plans for the test instances in ms-level time.

Table I: Performance of different algorithms

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.

We also monitor the use of groups and bins in the allocation plans. Fig. 2 shows the results and highlights the relationship between the number of groups and the number of bins; this illustrates the effect of the grouping function (A step 1). We show the results from large topologies (L-Tree, L-Guru, and L-Senti). For each topology, we show the best and worst results. As we can see, the packing results are worse when items are partitioned into more groups. This is because tasks that are partitioned into different groups will not be packed into the same processor. Let G'_g denote a group, then v_i can be put into G'_g if and only if $\forall v_j \in G'_g, h_j + t_i <= \bar{H}$. But two tasks in two groups are not necessarily backup dependent and there is a possibility that they can be packed into the same processor. As a result, more partition groups lead to much stricter constraints, and therefore, can waste more resource.



Note: $*$ indicates worst cases.

Figure 2: The effect of the number of groups.

V. Related work

The task allocation problem is one of the fundamental issues in distributed systems. Related works on stream processing systems propose models with the objectives of balancing workload [4] and minimizing latency [3], [5]. Related works have been focusing on the task allocation problem in a failure-free scenario that does not take the effects of failures into account. The proposed RTAP problem (as well as the related task allocation problem) is related to the Bin Packing (BP) problem. For the general BP problem and the two-dimensional BP problem, please refer to [16], [17]. The main challenge of RTAP is that the item heights (recovery latency) are related to both the partial packing solution (task allocation plan) and the stream topology.

Although Fault-Tolerant (FT) and reliability in distributed systems [18] have already been widely studied, SPM introduces new challenges [1]. Existing FT approaches enable stream processing systems to recover from task failures with a short recovery latency [7], [8], [10], [11], [13]. The upstream backup model [11] is one of the most widely applied approaches. However, it introduces recovery latencies. Recent research [8], [13], [14] studies FT strategies for stream processing applications with a focus on better resource utilization and recovery performance. In our previous work [8], a task-level failure effect model is proposed to make backup plans with minimum overhead and guaranteed task recovery latency. However, correlated task failures caused by task allocation plans are not considered. To the best of our knowledge, this is the first work to study the recovery latencies caused by correlated task failures in stream processing systems.

VI. Conclusions

This paper focuses on a task allocation strategy for distributed stream processing systems considering processor failure effects. We propose a novel, quantitative, correlated failure effect model to describe the relationship between the recovery latency and task allocation plans (the packing of tasks into processors) of a stream topology. We introduce the Recovery-latency-aware Task Allocation Problem (RTAP) based on the model and propose an approach to compute the task allocation plan with a recovery latency guarantee and a time complexity of $O(n \log^2 n)$. The proposed method is effective (using 15%-20% fewer resource compared to benchmarks) and efficient (using ms-level time), which makes it applicable to real production environments making both off-line task allocation decisions and on-line task reallocation decisions.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) (grant 61602205), by the Major Special Research Project of the Science and Technology Department of Jilin province (grant 20160203008GX), by the China Scholarship Council, and by the NSF of the U.S. (grants CNS 1629746, CNS 1564128, CNS 149860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, and ECCS 1231461).

REFERENCES

[1] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[2] G. Hesse and M. Lorenz, "Conceptual survey on data stream processing systems," in *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. IEEE, 2015, pp. 797–802.

[3] R. Eidenbenz and T. Locher, "Task Allocation for Distributed Stream Processing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[4] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applitions," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 69–80.

[5] B. Lohrmann, P. Janacik, and O. Kao, "Elastic stream processing with latency guarantees," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*. IEEE, 2015, pp. 399–410.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002, pp. 1–16.

[7] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[8] H. Li, J. Wu, Z. Jiang, X. Li, and X. Wei, "Minimum backups for stream processing with recovery latency guarantees," *IEEE Transactions on Reliability*, vol. PP, no. 99, pp. 1–12, 2017. DOI: 10.1109/TR.2017.2712563.

[9] H. Li, X. Wei, Q. Fu, and Y. Luo, "Mapreduce delay scheduling with deadline constraint," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 3, pp. 766–778, 2014.

[10] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "Timestream: Reliable stream computation in the cloud," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 1–14.

[11] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. IEEE, 2005, pp. 779–790.

[12] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch, "Integrating scale out and fault tolerance in stream processing using operator state management," in *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. ACM, 2013, pp. 725–736.

[13] L. Su and Y. Zhou, "Tolerating correlated failures in massively parallel stream processing engines," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 517–528.

[14] T. Heinze, M. Zia, R. Krahn, Z. Jerzak, and C. Fetzer, "An adaptive replication scheme for elastic data stream processing systems," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*. ACM, 2015, pp. 150–161.

[15] A. Salama, C. Binnig, T. Kraska, and E. Zamanian, "Cost-based fault-tolerance for parallel data processing," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 285–297.

[16] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of computer and system sciences*, vol. 9, no. 3, pp. 256–278, 1974.

[17] E. G. Coffman Jr, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: survey and classification," in *Handbook of Combinatorial Optimization*. Springer, 2013, pp. 455–531.

[18] J. Wu, *Distributed system design*. CRC press, 1998.