Mobile Networks and Applications (2021) 26:1950–1959
https://doi.org/10.1007/s11036-020-01729-7

Research on Optimal Checkpointing-Interval for Flink Stream Processing Applications

Zhan Zhang[1] · Wenhao Li[1] · Xiao Qing[1] · Xian Liu[1] · Hongwei Liu[1]

Accepted: 6 December 2020 / Published online: 6 January 2021

Abstract
Nowadays various distributed stream processing systems (DSPSs) are employed to process the ever-expanding real-time data. The DSPSs are highly susceptible to system failure, and the fault-tolerance issue is a major problem, which is getting lot of attention nowadays. Flink is a popular streaming computing framework that implements a lightweight, asynchronous checkpoint technique based on the barrier mechanism to ensure high efficiency in analysing the data. In a checkpoint-based fault-tolerance mechanism, a shorter checkpoint interval can increase runtime cost of streaming applications, while a longer one will increase recovery time of failure recovery. So, selecting an optimal checkpoint interval is critical to attain high efficiency of the streaming applications. Traditional optimal checkpoint interval mechanisms usually assume that the checkpointing delay and the fault recovery time are fixed. However, both factors have a strong relation to the intensity of the application's workload. To obtain more optimal checkpoint interval under different workload intensities, this paper proposes a performance model to estimate the tuples processing latency and a recovery model to estimate the fault recovery time. With these two models, an optimal checkpoint interval can be arrived. These models and the interval optimisation interval are verified experimentally on Flink. The results show that the proposed model can recommend an optimal checkpoint interval according to the system reliability related indicators. This proposed system optimised recovery time and performs efficiently in applications with delay constraints.

Keywords Stream processing · Fault-tolerance · Checkpoint interval · Queue model

Large amount of real-time data are generated in the network with the development of smart home, smart transportation, social media, Internet of things and Internet applications. These data are collected, processed and analysed in real-time, so that the data processing results can be delivered in real-time with sub-second delay. The requirement for processing these real-time data spawned the concept of stream computing. Stream computing is a paradigm of memory computing that runs in a distributed environment and is therefore highly susceptible to system failure [1].

Once the stream processing system fails, the system must be normalised or get everything back to normal as soon as possible. Otherwise, the final estimated result may be worthless due to timeout. The fault-tolerance mechanism of streaming computing systems is one of the important research fields, which has received a greater attention from academia and enterprises. So far, mainstream stream processing systems generally use active or passive backup to enhance their reliability. Active backup can realise seamless failure recovery by switching among active standby tasks, but doubles the resource consumptions. As a result the cost will be very high when used in large-scale applications. Because of the high costs of active backup, checkpoint-based passive backup mechanisms are getting very popular, which reduce the expenses.

Flink [2] is an open source stream processing framework for distributed, high-performance stream processing applications. Compared with other stream processing engines such as Storm [3] and Spark Streaming [4], Flink can support both stream processing and batch processing, support real-time data processing with better throughput and exactly-once semantics process. Moreover, it also supports application in flexible deployment and flexible

✉ Wenhao Li
liwenhoho@gmail.com

1 School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

expansion. Therefore, it is widely used in production environments. A comparative study is performed with existing distributed stream processing engines [5].

Flink implements a lightweight asynchronous checkpoint based on the barrier mechanism to ensure high availability and efficiency. Choosing an optimal checkpoint interval is critical for checkpoint-based stream processing systems to ensure efficiency of the streaming applications. A shorter checkpoint interval will increase the cost of the streaming applications. Moreover, a longer checkpoint interval will increase the failure recovery time. Therefore, consumers need to balance between the additional cost for trouble-free operation and the cost of failure recovery to get the best quality of service.

The frequent alignment of the barrier markers will have an impact on the data processing delay, especially under a heavy workload in the exactly-once semantics process. However, Flink requires users to determine the checkpoint interval prior to the deployment phase. Therefore the user may set an inappropriate checkpoint interval due to incorrect evaluation of the incoming load characteristics, which adversely affect the quality of service of the streaming application. To solve the above problems, we are proposing a new model and its scheme of steps are as follows:

(1) Performance model based upon Jackson's open queuing network is used to estimate the tuples processing latency with different workloads and checkpoint intervals.
(2) A recovery model is used to estimate the fault recovery time with different workloads and checkpoint intervals.
(3) Checkpoint interval optimisation method based on the above models is used to calculate an optimised checkpoint interval with the system failure rate.
(4) Finally, experiments are conducted on Flink to verify the effectiveness and efficiency of our models and checkpoint interval optimisation method.

1 Related work

Active backup and checkpoint-based passive backup are two fault-tolerance techniques widely used in distributed stream processing systems [6]. When failures occur, active backup requires at least one active backup instance to enable the task to switch from the primary task to its backup task. This ensures the shortest fault recovery time, but also incurs high resource consumptions. Therefore, in the past, active backup is only suitable for stream processing systems deployed in a small number of servers [7].

As the volume of stream data increases, the stream processing system is getting more and more complex, and the fault-tolerance mechanism based on active backup will become inefficient and may even be unavailable [8]. In contrast, resource efficiency can be significantly increased by using checkpoint-based fault-tolerance mechanisms. Therefore, in recent years, a large number of research studies analysed checkpoint-based fault-tolerance mechanisms used in distributed stream processing platforms. Sebepou et al. [9] introduced a state partitioning mechanism that supported incremental state checkpoints with minimal pausing of operator processing, thereby reducing the update cost of checkpoints. Zaharia et al. [10] proposed a new method to selectively write memory checkpoints on stable storage, thereby reducing the cost of saving checkpoints. Castro et al. [11] and Heinze et al. [12] proposed a checkpoint fault-tolerance method combined with upstream backup to reduce the recovery cost of global checkpoints. Su et al. [13] proposed a checkpoint fault-tolerance method that combined partial active backup with passive backup, thereby taking advantage of active backup to reduce rollback-recovery time. In summary, we can say that checkpoint-based fault-tolerance mechanisms are getting more and more attention in distributed stream processing systems.

When checkpoint-based fault-tolerance mechanisms are used, an optimal checkpoint interval is the key to ensure high efficiency of the submitted stream processing applications. The optimisation of checkpoint interval has been extensively studied in the field of high-performance computing. In 1973, Young et al. [14] used a first-order approximation to calculate an optimised checkpoint interval. In 2006, Daly et al. [15] extended the first-order approximation and proposed a high-order approximation scheme by taking into account the cost of failure recovery. Fernandez et al. [11] demonstrated the effect of checkpoint interval on delay and fault recovery time through experiments and proposed a method to set checkpoint interval based on estimated fault frequency and performance constraints. Liu et al. [16] and Jin et al. [17] further optimised the checkpoint interval setting of different fault distributions and execution scales based on some classical checkpoint interval optimisation methods. There are only few studies that explored and analysed optimal checkpoint interval setting in the field of stream computing. Liu Zhiliang et al. [18] designed a protocol for dynamic adjustment of checkpoint intervals in the sudden flow scenarios through coordination between upstream and downstream nodes. Zhuang et al. [19] proposed the Optimal Checkpoint Interval Model to dynamically alter checkpoint interval based on online workload.

The research in the past assumed that the checkpoint time is a stable value, and the checkpoint interval only affects the number of checkpoints. However, in a real system, the checkpoint time is directly related to the checkpoint interval. To obtain a more optimal checkpoint interval under different workload intensities, a performance model is proposed by us to estimate the tuples processing latency and a recovery model to estimate the fault recovery time.

2 Modelling and analysis

2.1 System model

In the stream processing system, each stream processing job can be abstracted as a directed acyclic graph (not considering the existence of a loop), which is recorded as G = <V, E>, where V represents the set of operators and E represents the direction of data flow between operators. Since each operator in the topology may have multiple parallel instances in actual operation, this paper uses the vector $\boldsymbol{n} = (n_1, n_2, \ldots, n_{|V|})$ to represent the parallelism of each operator, i.e., there are n_i instances of the operator V_i.

Flink use barrier markers to achieve consistent asynchronous checkpoints. The barrier markers are periodically injected into the input data streams by the JobManager of Flink, which are special signals to command the operators to save the state. These signals are pushed throughout the whole stream processing graph.

As shown in Fig. 1, when the operator task receives all barrier markers from its pre-tasks, it makes a snapshot of its current state and broadcasts the barrier marker to all of its successor tasks. Flink supports incremental snapshot and allows asynchronous state snapshots with low costs [20]. As noted above, to guarantee the correct results from exactly-once processing, the operator task must wait for all barrier markers from its pre-tasks to be aligned before saving the snapshot. The input connection whose barrier marker arrives earlier will be blocked, and the tuple will be cached. This phase is the alignment phase of the barrier markers and is a significant increase in overhead cost of checkpoint. So we divide the overhead cost of checkpoint into two parts: fixed overhead and alignment overhead.

2.2 Performance model based on Jackson network

When the checkpoint mechanism is not enabled, the arrival rate of tuples that arrive at operator V_i is denoted by λ_i and the processing rate of operator V_i is denoted by μ_i. We assume that both the inter-arrival time of external tuples and the service time of the operators are independently and identically distributed exponential random variables. It is also assumed that load

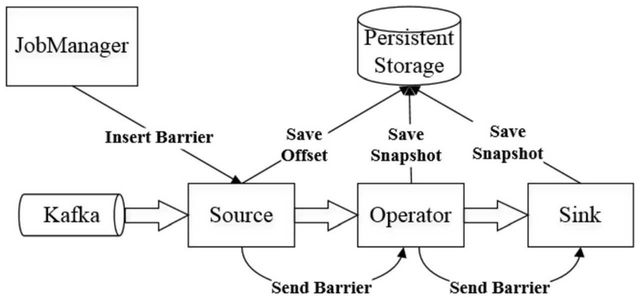


Fig. 1 Checkpoint Execution Flow Diagram

balancing is achieved in every operator. We need classify workload based on statistical features of traffic flows [21, 22]. Then, statistical tests are applied to check whether the load characteristics of the application conforms the distribution characteristics [23, 24].

The intensity of multiple inputs can be directly added, as the Poisson distribution is cumulative. Therefore, for each operator, it can be considered as conforming to the M/M/n queuing model, and the entire stream topology is modelled as the open Jackson queuing network. When the system reaches a stable state, the average sojourn time E[T_i] of the operator V_i consists of two parts: (1) the expected queuing delay, denoted by E[Q_i](M/M/n_i) and (2) the expected processing time, which is equal to $1/\mu_i$. Therefore, E[T_i] can be calculated using Eq. 1.

$$\mathrm{E}[T_i](n_i) = \mathrm{E}[Q_i](\mathrm{M/M}/n_i) + \frac{1}{\mu_i} \tag{1}$$

According to the Erlang delay formula, the expected queuing delay in the M/M/n service is calculated by Eq. 2.

$$\mathrm{E}[Q_i](\mathrm{M/M}/n_i) = \begin{cases} \dfrac{\pi_0 (n_i\rho_i)^{n_i}}{n_i!(1-\rho_i)^2 \mu_i n_i}, \rho_i < 1 \\ +\infty, \rho_i \geq 1 \end{cases} \tag{2}$$

In the above formula, $\rho_i = \frac{\lambda_i}{n_i}\mu_i$ denotes the resource utilisation of the operator V_i. It is easy to know that in the case of $\rho_i \geq 1$, the processing rate cannot keep up with incoming workloads, and the average tuple queuing delay will increase indefinitely. The probability that there is no tuple in the system under steady state is denoted by π_0, which can be calculated by Eq. 3.

$$\pi_0 = \left(\sum_{l=0}^{n_i-1} \frac{(n_i\rho_i)^l}{l!} + \frac{(n_i\rho_i)^{n_i}}{n_i!(1-\rho_i)} \right)^{-1} \tag{3}$$

Based on the theoretical model of the open Jackson queuing network, the average processing delay E[T]($\boldsymbol{n}$) of the entire queuing network can be obtained by adding the average processing delay E[T_i](n_i) of each operator, given by:

$$\mathrm{E}[T](\boldsymbol{n}) = \frac{1}{\lambda_0} \sum_{i=1}^{|V|} \lambda_i \mathrm{E}[T_i](n_i) \tag{4}$$

When a checkpoint is enabled in Flink, the probability distribution of the tuple arrival interval and the operator processing time does not change. According to the analysis in Section 2.1, the Flink save state snapshot to persistent storage can be performed asynchronously, so that the actual backup overhead is small, and the average delay of backup overhead is recorded as t_c. However, if there are multiple upstream operators for an operator, to guarantee exactly-once semantic it must wait for barriers from all upstream operators before taking snapshot of its current state asynchronously. This kind of barrier alignment operation will cause the tuples to arriving earlier to wait in queue, which will have a great impact on the overhead of the tuple latency. Therefore, the effect of the alignment operation on E[Q_i](M/M/n_i) will be modelled in the next section.

According to Flink's checkpoint mechanism, it can be concluded that: when the system reaches a stable state and if the overhead of the checkpoint alignment phase of the sub-instance O_i is Δi, then the tuple flowing to O_i waits for the checkpoint operation not to exceed Δe [25]. In this paper, the correctness of the results is proved only by the existence of two different upstream instances. When there are more than two upstream instances, similar analysis can be performed.

As shown in Fig. 2, the two upstream tuple flow of the operator instance O_i are s_1 and s_2, which are Poisson processes with parameters λ_1 and λ_2, respectively. We assume that the barrier of s_1 reaches O_i before s_2, and then O_i enters the alignment phase earlier. Obviously, within the aligned interval Δe, O_i caches approximately $\lambda_1 * \Delta$t tuples from s_1, denoted as $(e_1\ , e_2, \ldots e_{|\lambda_1 \Delta t|})$, where e_1 is the first tuple cached in the s_1 buffer queue. At the end of the alignment phase, O_i immediately continues to pass the barrier downstream, saves the current snapshot asynchronously, and processes the tuples in the buffer queues. It is easy to know that e_1 arrives first, and the waiting time is increased to Δt. If a subsequent tuple e_i enters the buffer queue by Δl later than e_1, then e_i actually waits for only ($\Delta t - \Delta l$) in the alignment phase. However, in the stage of processing the cache tuple, e_i needs to wait for the preceding ($\lambda_1 * \Delta l$) tuples to be processed initially. The front ($\lambda_1 * \Delta l$) tuple processing delay is approximately $\frac{\lambda_1 * \Delta l}{\mu_i} < \frac{\lambda_i * \Delta l}{\mu_i} \leq \Delta l$. Therefore, e_i needs to wait $\left(\Delta t - \Delta l + \frac{\lambda_1 \Delta l}{\mu_i}\right)$, which is smaller than Δt.

It is assumed that the overheads of the n_i instances of the operator V_i for the checkpoint alignment operation are t_1, t_2,… and t_{n_i}, respectively. Figures 1 to 3 shows the increment of the tuple wait time during one of the operator alignments. Based on above conclusions, we can say that the average

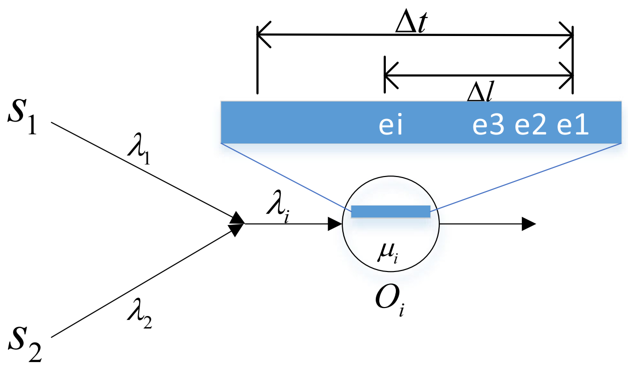


Fig. 2 Task level queuing delay analysis

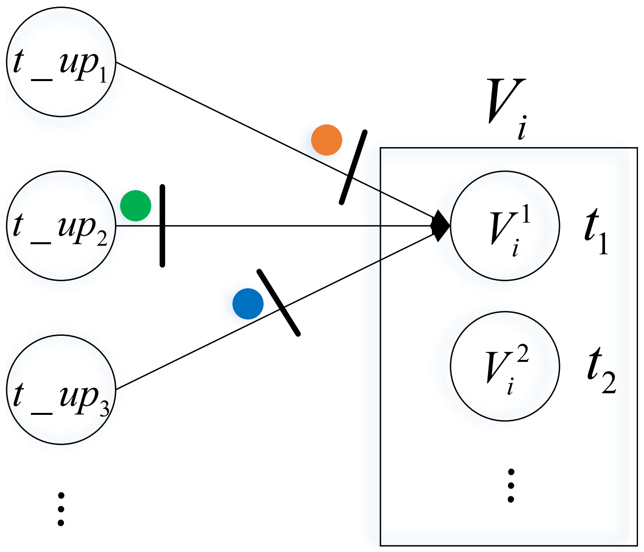


Fig. 3 Operator level queuing delay analysis

additional waiting time of all tuples cached in buffer queues does not exceed the average additional waiting time of the tuples cached in their respective buffer queue, which is $\frac{\sum_{k=1}^{n_i} t_k}{n_i}$, in the same operator.

It is known that$\mathrm{E}\left[align_{V_i}\right]$ represents the average additional wait time of the tuples caused by the barrier alignments and $\mathrm{E}\left[align_{V_i}\right]$ (interval) a function of the checkpoint interval. Tuples in each operator require approximately $\mathrm{E}[Q_i]\frac{(\mathrm{M/M}/n_i)}{interval}$ alignments in the M/M/n queuing model so that $\mathrm{E}\left[align_{V_i}\right]$ (interval) can be estimated according to Eq. 5.

$$\mathrm{E}\left[align_{V_i}\right](\mathrm{interval}) = \frac{\left(\frac{\pi_0 (n_i \rho_i)^{n_i}}{n_i!(1-\rho_i)^2 \mu_i n_i}\right) * \frac{\sum_{k=1}^{ni} t_k}{n_i}}{interval} \tag{5}$$

In summary, once the checkpoint mechanism is enabled in Flink, E[Q_i](M/M/n_i) can be approximated using Eq. 6.

$$\mathrm{E}[Q_i](\mathrm{M/M}/n_i) = \begin{cases} \frac{\pi_0 (n_i \rho_i)^{n_i}}{n_i!(1-\rho_i)^2 \mu_i n_i} + \mathrm{E}\left[align_{V_i}\right] + t_c, \rho_i < 1 \\ +\infty, \rho_i \geq 1 \end{cases} \tag{6}$$

2.3 Checkpoint recovery model

JobManager and TaskManager are the two main daemons (computer programs that run in the background) of the system. JobManager is responsible for both scheduling and failure recovery. The JobManager monitors the heartbeat information of each TaskManager through the Actor System to detect whether a fault has occurred or not. One can inspect or analyse the entire process of Flink fault recovery by tracing the log information of Flink.

When the target job is running normally, the subtasks in the job are in the running state. When the job implement the checkpoint and the restart policy, the JobManager detects the failure, the status of the job changes to Failing, and the status of each task in the job is switched to the cancelling or cancelled states. The JobManager will restart the job, and the Checkpoint Coordinator in the JobManager will restore the state of each task from the most recently completed checkpoint. Then, each subtask in the job is rescheduled and dispatched to a new slot and switched to the running state. At this point, JobManager should have restored the failed job to the state of the latest checkpoint. The job requires the reprocessing the records from Kafka, starting from the offsets that were stored in the checkpoint.

It is learned from the above failure recovery process that the overhead of the checkpoint recovery $T_{recovery}$ mainly consists of the following four parts: the fault detection time T_{detect}, the state recovery time $T_{restore}$, the task restart time $T_{restart}$ and the tuple reprocess time $T_{reprocess}$.

$$T_{recover} = T_{detect} + T_{restore} + T_{restart} + T_{reprocess} \tag{7}$$

The fault detection time depends mainly on the heartbeat timeout period and the value is set according to the actual needs. The state recovery time depends mainly on the size of the checkpoint state and the network bandwidth. Please note that, the checkpoint status does not exceed the size 10 MiB in our experiment and so the state recovery time is negligible. The time of the task restart is related to the experimental platform and the application topology scale, and the tuple reprocess time depends mainly on the number of tuples to be replayed, which is closely related to both the length of the checkpoint interval and the intensity of workload. This paper assumes that the workload intensity is λ. In a pipeline-style job, the throughput of the job depends on the least throughput operator in the job, which is recorded as $\min\{\mu_i\}$. In general, the number of reprocessed tuples can be expressed as $\lambda *$ interval/2, so $T_{reprocess}$ can be approximated by Eq. 8.

$$T_{reprocess} = \frac{\lambda * interval}{2 * \min\{\mu_i\}} \tag{8}$$

2.4 Optimised checkpoint interval method

When configuring the checkpoint interval, both the overhead of completing the checkpoint and fault recovery time should be considered. To make a trade off, these two parts are given weights to evaluating indicator F, where f_rate represents the failure rate of the cluster. A checkpoint interval that allows F to take a minimum value can be approximated based on Eq. 9.

$$F = \alpha * \mathrm{E}[T](\boldsymbol{n}) + (1-\alpha) * f\_rate * T_{restart} \tag{9}$$

In the above formula, the weight α has a value range of (0, 1). When the value of the weight α tends to 1, it indicates that the user is more concerned with the application performance. When the weight α tends to 0, it indicates that the user is more concerned with the application recovery time.

3 Experimental results and analysis

3.1 Experimental environment

To verify the accuracy of the checkpoint performance model and the checkpoint interval optimisation model in the actual system, several experiments are performed on both the simple topology and the complex topology. This article uses Apache Flink as the stream processing platform, Apache Kafka as the intermediate message queue, Apache Zookeeper as the distributed application coordination service and Apache Hadoop and RocksDB as the state backstage to save the snapshot state. The software and hardware experimental environment are shown in Tables 1 and 2, respectively. The experimental topology is shown in Fig. 4. In our experiments, the input uses simulated data to approximate the negative exponential distribution with a parameter λ. The operator processing logic is replaced with a processing delay time, and the processing delay time is approximated to a negative exponential distribution with the parameter μ.

3.2 Experiments and analysis

3.2.1 Checkpoint performance model verification

Now the accuracy of the checkpoint performance model is verified with the set of experiments. So, the experiments that may shed light on the relationship between the processing delay of the tuple and the checkpoint interval under different resource utilisation rates were performed. The processing time of the tuples were subjected to negative exponential distribution with parameters μ of 100, 300 and 500, respectively. Tuples were run five times on the cluster and the average tuple latency was recorded for the five experiments under different

Table 1 Cluster hardware configuration

Hardware	Configuration
CPU	Intel(R) Xeon(R) CPU E5–2620 v2 @ 2.10GHz * 2
Memory	128GB
Disc	2 TB HDD
NIC	1000 Mbps

Table 2 Software configuration

Software	Version	Number of Instances
OS	CentOS 7	9
Flink	1.7.2	1JobManager 5TaskManager
Kafka	2.12–0.11.0.3	3 Brokers
Zookeeper	3.4.13	3
Hadoop	2.8.5	3

experimental parameter configurations. Since it is difficult to accurately control the parameters such as λ and μ in the Flink system, these parameters also need to be adjusted according to the actual cluster. Figures 5 and 6 show observed and theoretical results of simple topology and complex topology, respectively, when the parallelism is 4 and 1.

As shown in Fig. 5, the Y-axis represents the tuple average latency under different workload intensity and operator utilisation, while X-axis represents the checkpoint interval. The curve labelled 'Measured' represents the real tuple average latency measured during the experiment, while the curve labelled 'Estimated' is the tuple average latency calculated by the checkpoint performance model. The experimental results show that the tuple average latency shows a gradual decline with the gradual increase of the checkpoint interval under different workload intensity λ and resource utilisation ρ. Moreover, when the checkpoint interval is short, the latency decreases more rapidly and the amplitude is larger. For each increase in the checkpoint interval, the latency is reduced by approximately 1% to 10%; however, as the checkpoint interval continues to increase, the trend of latency decline also slows down significantly.

The measurements for the complex topology in Fig. 6 show similar characteristics. It should be noted that the effect of checkpoint operations on tuple processing delays in this model depends on not only the checkpoint interval, but also the time of a checkpoint alignment. In the complex topology experiment, as the parallelism of the topology is set to 1, there will be only one alignment operation at the join operator, and the checkpoint interval has less influence on the average processing delay. By comparing the theoretical and experimental results, It can be concluded that the observed data conform to the data estimated by our performance model.

3.2.2 Checkpoint recovery model verification

Experiments are conducted to verify the checkpoint recovery model. In these experiments faults are injected into the TaskManager to trigger Flink fault-tolerance mechanism and a mass of data under different conditions are collected. As

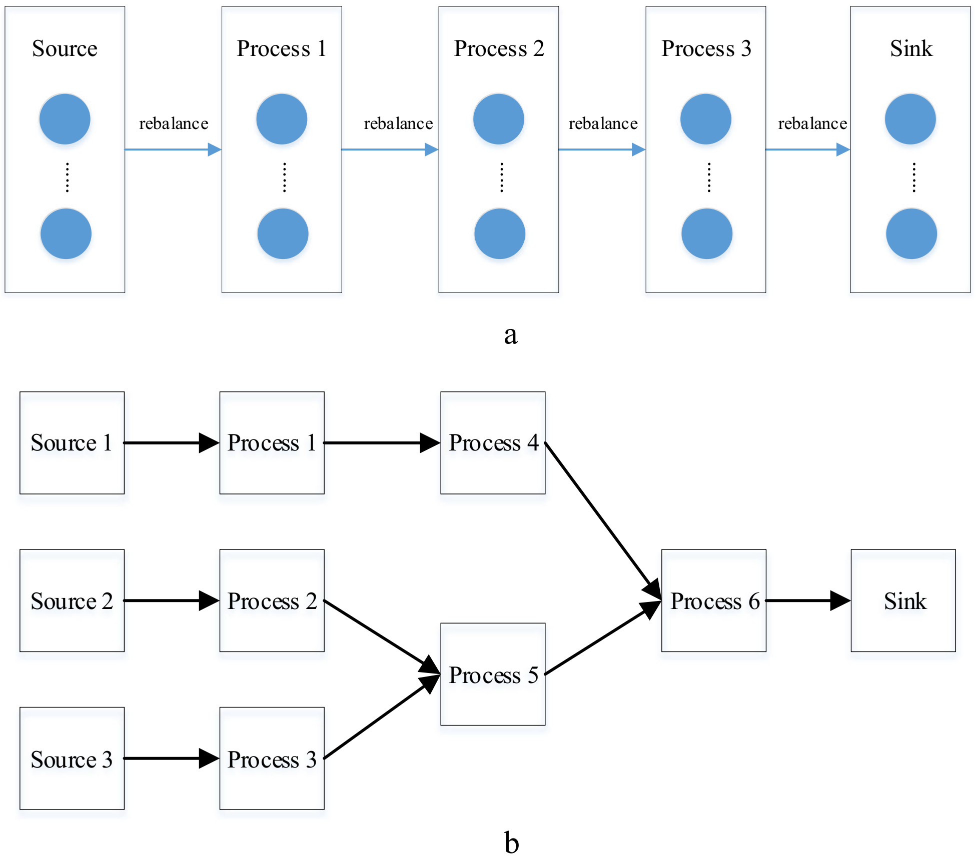
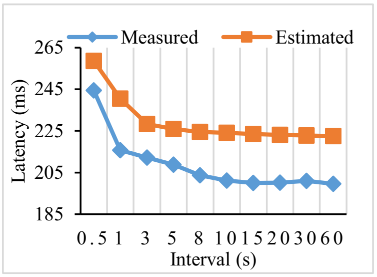
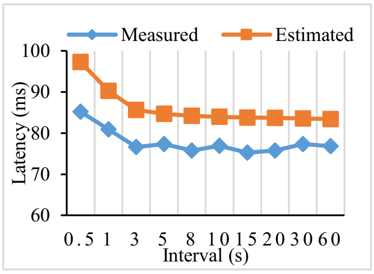


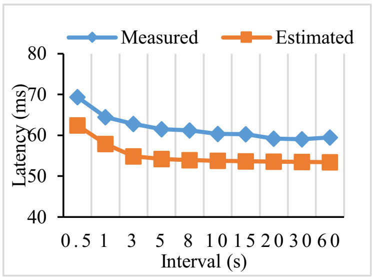
Fig. 4 Experimental topology



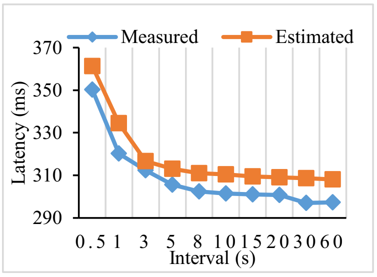
(g) $\mu = 100$, $\rho = 0.85$



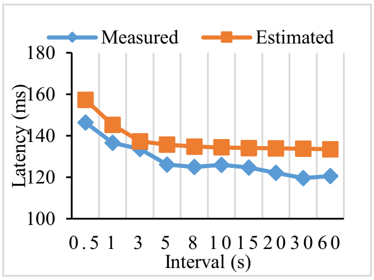
(h) $\mu = 300$, $\rho = 0.85$



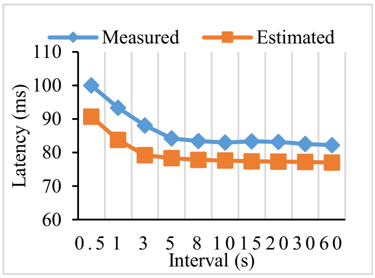
(i) $\mu = 500$, $\rho = 0.85$



(j) $\mu = 100$, $\rho = 0.9$



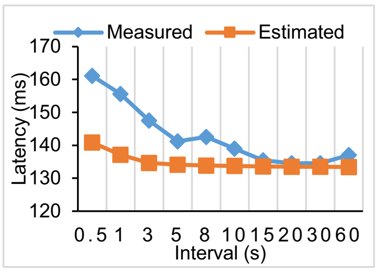
(k) $\mu = 300$, $\rho = 0.9$



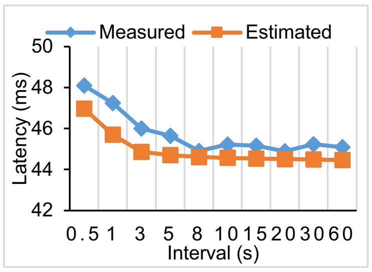
(l) $\mu = 500$, $\rho = 0.9$

Fig. 5 Relationship between latency and checkpoint interval of pipeline topology

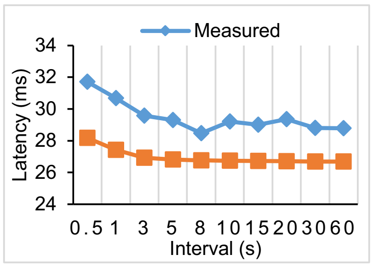
shown in Fig. 7, the Y-axis represents the reprocess time under different operator utilisation, while X-axis represents the checkpoint interval. The bars labelled 'Measured' represent the real reprocess time measured during the experiment under different workload intensity, while the bars labelled 'Estimated' represents the reprocess time calculated by the checkpoint recovery model. In these experiments, the service time of the operators are homogeneous. Therefore, It can be seen from Eq. 8, the reprocess time is only related to checkpoint interval and operator utilisation. The results prove this conclusion and show that under different checkpoint intervals, the average reprocess time measured by multiple experiments are very close to the estimated reprocess time of our model.



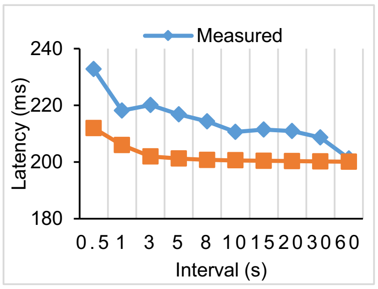
(a) $\mu = 100$, $\rho = 0.85$



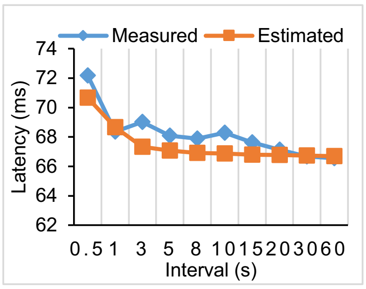
(b) $\mu = 300$, $\rho = 0.85$



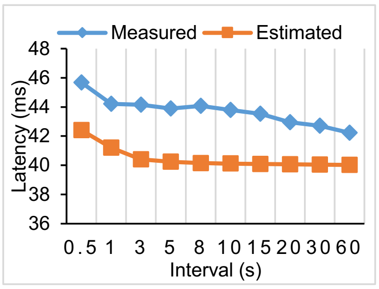
(c) $\mu = 500$, $\rho = 0.85$



(d) $\mu = 100$, $\rho = 0.9$



(e) $\mu = 300$, $\rho = 0.9$



(f) $\mu = 500$, $\rho = 0.9$

Fig. 6 Relationship between latency and checkpoint interval of complex topology

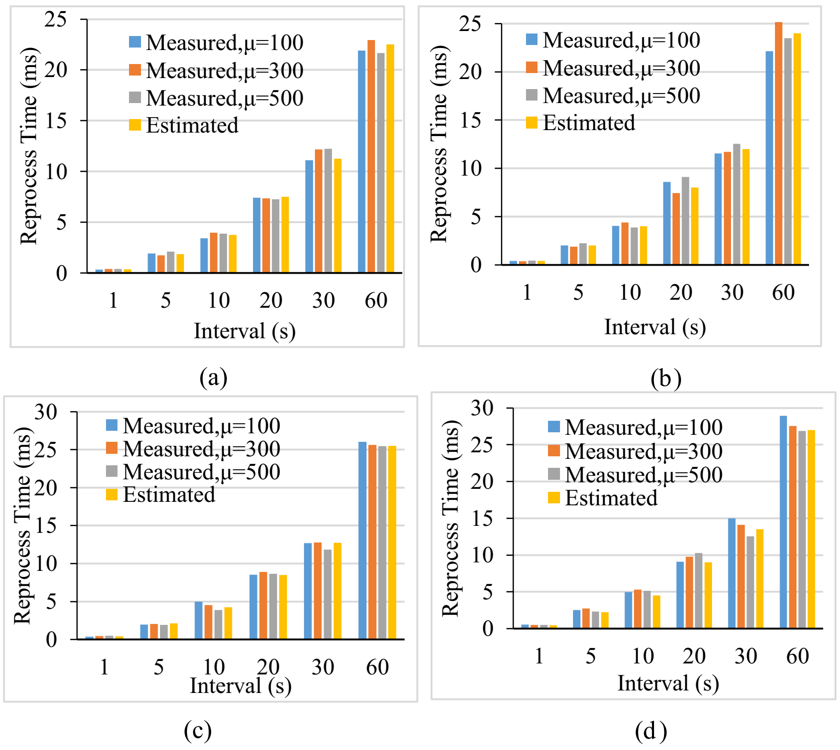


Fig. 7 Relationship between reprocessing time and checkpoint interval

3.2.3 Optimal checkpoint interval method validation

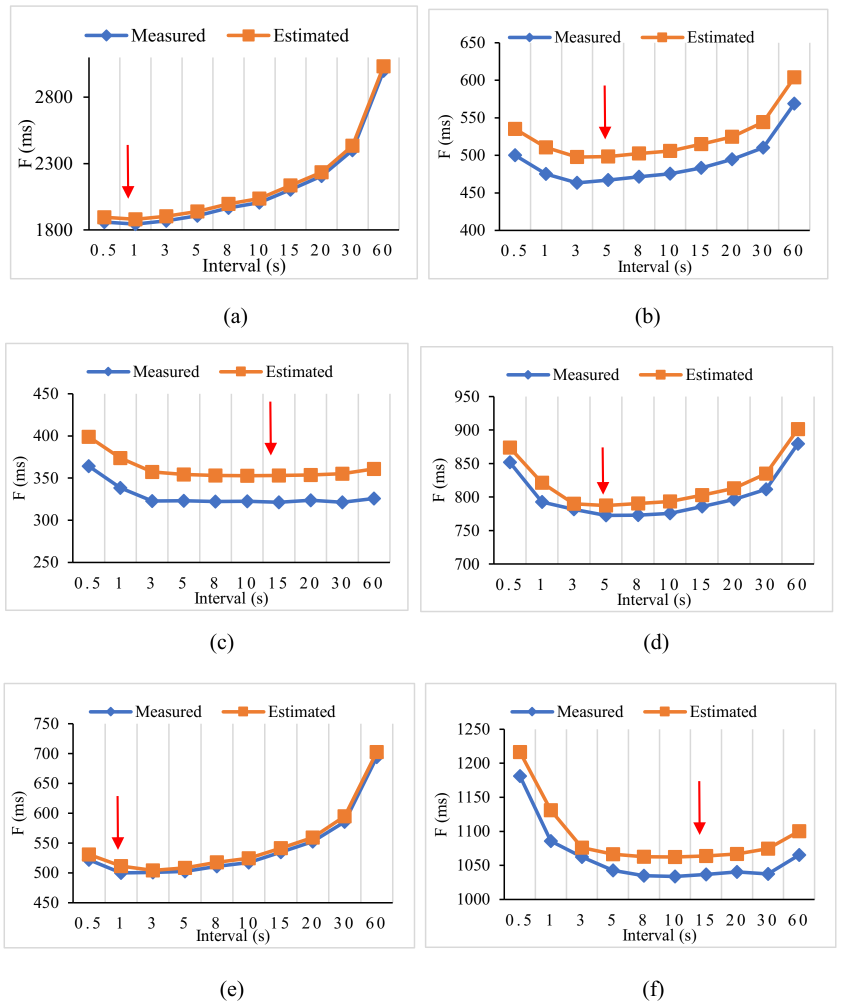
In this set of experiments, the optimal checkpoint interval recommended based on F is observed by adjusting the failure rate f_rate and the weight α. $F = \alpha * \mathrm{E}[T](\boldsymbol{n}) + (1-\alpha) * f\_rate * T_{restart}$ In the above formula, When the value of the weight α tends to 1, it indicates that the user is more concerned with the application performance. When the weight α tends to 0, it indicates that the user is more concerned with the application recovery time. According to the actual situation of the cluster, except the reprocess time, the time required for fault detection, state recovery and topology restart, is about 30s. In Fig. 8, the red arrows indicate the recommended optimal checkpoint interval.

The experimental results show that when the failure rate is small (such as 0.1%), the minimum value of F approaches to longer checkpoint interval; when the failure rate increases gradually (increases to 10%), the minimum value of F approaches to shorter checkpoint interval. It can be observed by comparing Fig. 8b and d that the recommended checkpoint interval increases with the increase in input rate λ for the same failure rate. As λ increases, the checkpoint interval has a significant impact on the tuple processing delay, especially when the utilisation ρ is close to 1. The influence of the weight α on the recommended checkpoint interval can be observed through analysing the Fig. 8d, e and f. When α tends to 0, the user is more concerned about the fault recovery time. At this time, the recommended checkpoint interval will be shorter to guarantee rapid recovery when a fault occurs; when α tends to 1, the user is more concerned about the impact of the checkpoint on the processing delay. At this time, the recommended checkpoint interval will increase accordingly. In summary, based upon the performance of different application parameters and results, we can conclude that the proposed checkpoint interval optimisation model system is a reliable one and has better user requirements and can be recommended for getting a better checkpoint interval.

4 Conclusion

This paper analyses the fault-tolerance mechanism of Flink lightweight asynchronous checkpoint and its performance based on Jackson queuing network. The experimental results show that our proposed checkpoint optimisation performance model based on the Jackson network is similar to the Flink system processing latency as the checkpoint interval changes, and the deviation between the estimated and measured values does not exceed 15%. The statistical results of the checkpoint recovery model in multiple experiments are very close to the estimated values. The F optimisation indicator proposed in our paper can recommend a more appropriate checkpoint interval depending on the system's failure rate and the application's operating parameters. Experiments show that the checkpoint interval recommendation calculated by our checkpoint interval optimisation model is consistent with the checkpoint interval recommendation obtained by using actual values.

Fig. 8 The calculation of the optimised checkpoint interval



In the future, we have planned to analyse different types of practical application topologies and examine how checkpoint interval can be adjusted based on real-time varying workloads.

References

1. Akber SMA, Chen H, Wang Y, Jin H (2018) Minimizing overheads of checkpoints in distributed stream processing systems. In 2018 IEEE 7th international Conference on Cloud Networking (CloudNet) (pp. 1-4). IEEE
2. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink: stream and batch processing in a single engine. Bull IEEE Comp Soc Tech Committee Data Eng 36(4):28–38
3. Iqbal MH, Soomro TR (2015) Big data analysis: apache storm perspective. International journal of computer trends and technology 19(1):9–14
4. Chintapalli S, Dagit D, Evans B, Farivar R, Graves T, Holderbaugh M, ..., Poulosky P (2016) Benchmarking streaming computation engines: Storm, flink and spark streaming. In 2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW) (pp. 1789–1792). IEEE
5. Lal DK, Suman U (2019) Towards comparison of real time stream processing engines. In 2019 IEEE Conference on Information and Communication Technology (pp. 1-5). IEEE
6. Hwang JH, Balazinska M, Rasin A, Cetintemel U, Stonebraker M, Zdonik S (2005) High-availability algorithms for distributed stream processing. In 21st International Conference on Data Engineering (ICDE'05) (pp. 779-790). IEEE
7. Balazinska M, Balakrishnan H, Madden S, Stonebraker M (2005) Fault-tolerance in the borealis distributed stream processing system. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data (pp. 13-24)
8. Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, ..., Bhagat N (2014) Storm@twitter. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data (pp. 147–156)
9. Sebepou Z, Magoutis K (2011) CEC: Continuous eventual checkpointing for data stream processing operators[C]. 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). IEEE, 145–156
10. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. HotCloud 10(10–10):95
11. Castro Fernandez R, Migliavacca M, Kalyvianaki E, Pietzuch P (2013) 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 (pp. 725-736)

12. Heinze T, Zia M, Krahn R, Jerzak Z, Fetzer C (2015) An adaptive replication scheme for elastic data stream processing systems. In: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (pp. 150-161)
13. Su L, Zhou Y (2017) Passive and partially active fault tolerance for massively parallel stream processing engines. IEEE Trans Knowl Data Eng 31(1):32–45
14. Young JW (1974) A first order approximation to the optimum checkpoint interval. Commun ACM 17(9):530–531
15. Daly JT (2006) A higher order estimate of the optimum checkpoint interval for restart dumps. Futur Gener Comput Syst 22(3):303–312
16. Liu Y, Nassar R, Leangsuksun C, Naksinehaboon N, Paun M, Scott SL (2008) An optimal checkpoint/restart model for a large scale high performance computing system. In 2008 IEEE International Symposium on Parallel and Distributed Processing (pp. 1-9). IEEE
17. Jin H, Chen Y, Zhu H, Sun XH (2010) Optimizing HPC fault-tolerant environment: an analytical approach. In 2010 39th International Conference on Parallel Processing (pp. 525-534). IEEE
18. Zhiliang L (2017) Research on adaptive checkpoint mechanism for large-scale streaming data processing. (Doctoral dissertation)
19. Zhuang Y, Wei X, Li H, Wang Y, He X (2018) An optimal checkpointing model with online OCI adjustment for stream processing applications. In 2018 27th International Conference on Computer Communication and Networks (ICCCN) (pp. 1-9). IEEE
20. Vianello V, Patiño-Martínez M, Azqueta-Alzúaz A, Jimenez-Péris R (2018) Cost of fault-tolerance on data stream processing. In: European Conference on Parallel Processing (pp. 17-27). Springer, Cham
21. Sun G, Chen T, Su Y, Li C (2018) Internet traffic classification based on incremental support vector machines. Mobile Networks and Applications 23(4):789–796
22. Sun G, Li J, Dai J, Song Z, Lang F (2018) Feature selection for IoT based on maximal information coefficient. Futur Gener Comput Syst 89:606–616
23. Feitelson DG (2015) Workload modeling for computer systems performance evaluation. Cambridge University Press, Cambridge
24. Stephens MA (1974) EDF statistics for goodness of fit and some comparisons. J Am Stat Assoc 69(347):730–737
25. Xiao Q (2019) Research on fault-tolerant strategy optimization for Flink stream processing framework [master's thesis]. Harbin Institute of Technology, Harbin