Chapter 6

Evaluation

This chapter presents the experiments devised to evaluate the advantages of employing the SIC quality metric in a stream processing system. All experiments were performed using the DISSP prototype implementation described in Chapter 4. The research questions that this chapter addresses aim at a better understanding of the characteristics of the SIC metric and its applicability in the context of a constantly overloaded stream processing system. The SIC metric was designed to be operator agnostic, so that it could be employed in a general purpose system supporting any type of query, but was intended to be used as an indicator about the quality of the computed results. Does the SIC value reflect the correctness of the delivered results? What is the correlation between the error of the output and the reduction in SIC values? The SIC metric is thought to be helpful in the implementation of semantic shedding policies. A fair shedding policy was designed and the experiments compare its performance to a random shedder. Does this policy achieve a better quality for the results? Is it more fair in the allocation of system resources among queries? We were also interested in evaluating the performance of this fair shedding policy in terms of scalability, both when varying the number of nodes and the number of queries deployed. What is the behaviour of this fair shedder, when the amount of processing resources changes? Does the performance scale linearly with the number of processing nodes? What happens when the number of queries is increased on the same processing infrastructure? A user may be willing to trade some correctness of result with a reduction of costs, deploying a larger number of queries on the same processing infrastructure, if the reduction in cost is proportionally greater than the reduction in SIC value. Is it possible to strike a trade-off between the achieved SIC value and the cost of renting the processing infrastructure? Loading the system with an increasing number of queries reduces the cost per query. How does this reduction in cost correlate with the reduction in SIC values? All these questions are answered in this chapter, by looking at a range of sample queries using both synthetic and real-world input data.

6.1 Experimental Setup

This section describes the experimental setup used in this chapter. We use two testbed: a local one and one on Emulab, as described in Table 6.1. The local testbed consists of 3 machines with 1.8 Ghz CPUs and 4 GB of memory running Ubuntu Linux 2.6.27-17-server. They are connected over a 1 Gbps network. One machine is used as an oracle, with a global system view, one for the input sources and query submission and one as a processing node. The Emulab testbed consists of a varying number of pc3000-type machines connected over a 1 Gbps LAN network. Each machine has a 3 Ghz CPU, 2 GB of memory and runs the FBSD410+RHL90-STD Emulab-configured Linux image. One machine is used as an oracle, three as input sources and three for the submission of queries.

The workload chosen for the experiments belong to two query classes: aggregate (i.e. AVG and COUNT) and complex (i.e. TOP-5, AVG-all and COV) queries. They are summarised in Table 6.2. The queries use a diverse set of operators, namely: average, top-k, group-by, filter, join, covariance, time-window, remote-sender, remote-receiver and output. The first query class consists of two aggregate queries, chosen to investigate the behaviour of the SIC metric under the different operator semantics. The second class consists of more complex queries, used to explore the properties of the SIC metric in workloads employing a broader variety of operators and outside the aggregate domain. These query types are used in a variety of data processing applications, such as sensor networks and social media analysis.

The input data generated by the sources contains either real-world load measurements or a spe-

Local Testbed	
Physical configuration	3 machines with 1.8 Ghz CPUs and 4 GB of memory running Ubuntu Linux 2.6.27-17-server and connected a over 1 Gbps network.
System layout	1 machine: oracle node, 1 machine: source data generation and query submission, 1 machine: DISSP processing node.
Data sources	400 tuples/sec in 5 batches/sec of 80 tuples/batch.
Emulab Testbed	
Physical layout	pc3000-type machines connected over a 1 Gbps LAN network. Each machine has a 3 Ghz CPU, 2 GB of memory and runs the FBSD410+RHL90-STD Emulab-configured Linux image.
System layout	1 machine: 1 oracle node, 3 machines: source data generation, 3 machines: query submission, 18 machines: 18 DISSP processing nodes.
Data sources	150 tuples/sec in 3 batches/sec of 50 tuples/batch.

Table 6.1: Testbed configurations for experiments.

Aggregate Queries	
AVG	Calculates the average value over 1 sec.
COUNT	Counts the number of tuples with values >= 50.0 over 1 sec.
Complex Queries	
TOP-5	Shows the 5 PlanetLab nodes with the highest amount of available CPU and at least 100 KB of free memory over 1 sec.
COV	Shows the covariance of the CPU consumption between two PlanetLab nodes.

Table 6.2: Query workload for experiments.

Synthetic source data	
Gaussian	Gaussian data distribution of mean 50.
Uniform	Uniform data distribution of mean 50.
Exponential	Exponential data distribution of mean 50.
Mixed	Random mix from the gaussian, uniform and exponential input sets.
Real-world source data	
PlanetLab	CPU and memory measurements collected on PlanetLab in April 2010 by the CoTop project.

Table 6.3: Input source data for experiments.

cific synthetic workload. The real-world data streams are traces from CPU load and memory usage measurements from all PlanetLab nodes [PLB12] collected in April 2010, as recorded by the CoTop project [CoD10]. The values of the synthetic data follow either a *gaussian*, *uniform* or *exponential*, as described in Table 6.3. Furthermore, a *mixed* synthetic workload is used that mixes all three distributions randomly. These synthetic workloads provide controlled experimental input sets that allow for an easier understanding of the results. In all experiments, the duration of the source time window (STW) is set to 10 seconds for all sources. This value stays well within the variation of processing delays of all the chosen queries. The shedding interval is set to 250 milliseconds, a value that provides a good trade-off between throughput and management overhead.

In the experiments using more than one processing node, comparison and scalability, the deployment of queries tries to emulate the scenario of a federated resource pool, in which the allocation of resources is not homogeneous. Each query is partitioned into a number of subqueries and these fragments are deployed on the available node according to the Zipf's distribution [AH02]. Consider, for instance, a scenario in which the number of nodes is 20, divided into 4 clusters of 5 nodes each, with a total

number of query partitions of 150. According to the Zipf's law, each cluster is assigned twice as many subqueries as the previous one. Starting with the deployment of 10 partitions onto the first cluster, the second would receive 20, the third 40 and the fourth 80.

6.2 SIC Values and Correctness

First, we evaluate the correlation between the SIC values of the output tuples and the correctness of the computed results, across a representative range of *aggregate* and *top-k* classes of queries as described in Table 6.2. The experiments use as input synthetic workloads that contains inputs arranged according to some well-known statistical distributions as well as real load measurements collected on PlanetLab, as described in Table 6.3.

For this set of experiments the local testbed is used, as described in Table 6.1. A single DISSP processing node is overloaded by instantiating an increasing number of queries of one class. As we increase the number of queries, the node is forced to discard an increasing number of input tuples. The node uses a load-shedder that drops tuples randomly from each query. The experiment is repeated for every query class and across the five different sets of source data.

Correlation Metrics

We compare the results of a query with degraded processing (i.e. with result SIC values of less than 1) against the result of a query with perfect processing. Each query, degraded and perfect, runs for 5 minutes and the error in the results is measured every second as a function of the achieved SIC value. For each query class, the experiment is repeated for all the different input data sets.

For the AVG and COUNT aggregate queries, we use the *Mean Absolute Error* (MAE) to quantify the relative distance of the *degraded* from the *perfect* result value across all measurements for the duration of the experiment:

$$\text{MAE} = \frac{1}{n} \sum \left| \frac{degraded - perfect}{perfect} \right| \tag{6.1}$$

For the TOP-5 query we calculate the error using the Kendall distance metric [FKS03] that counts the differences (i.e. permutations and elements in only one list) of pairs of distinct elements between the two lists. The Kendall's distance (τ) is defined as:

$$\tau = \frac{C_p - D_p}{\frac{1}{2}n(n-1)} \tag{6.2}$$

where C_p is the number of concordant pairs, D_p is the number of discordant pairs and n is the total number of pairs. This value is normalised to lie within the [0,1] interval.

In the case of the COV query, we compare the standard deviation of the real covariances with the one under overload. The real covariance come from perfect processing and this value is matched with the covariance obtained for degraded processing. Hence, we can evaluate the correlation between the SIC values and the quality of the COV query results.

Experimental Results

The following graphs present the results from the experiments running queries belonging to the *aggregate* and *complex* classes. A graph is shown for each query type (AVG, COUNT, TOP-5 and COV). Each graph contains the measurements obtained from the different runs of the experiment, one for each input data set.

Aggregate queries. For the AVG queries (see Figure 6.1), the graph shows a small error even under heavy overload. This is due to the particular nature of the average operation. Since the load-shedder discards tuples at random, the distribution of the data is not significantly affected. However, we observe that as the amount of load-shedding diminishes and the SIC values increase, the degraded values are closer to the perfect ones.

The COUNT query (see Figure 6.2) shows a different behaviour. The error grows linearly with the amount of discarded data. This is an extreme case, in which the occurrence of load-shedding has always a direct effect on the final results. Each tuple that is discarded, in fact, reduces the total output value (i.e. the final count) and thus increases the error.

Complex queries. For the TOP-5 query (see Figure 6.3), the error expressed as the Kendal distance decreases almost linearly with the amount of load-shedding performed, showing a good correlation between the SIC metric and the correctness of results for this type of queries.

For the COV query (see Figure 6.4), the standard deviation of results shows a decreasing trend as the amount of load-shedding diminishes and the SIC value increases. This means that when the amount of overload is reduced, the spread in the value for the results is also reduced. All input distributions, real and synthetic, show a low error despite the large amount of load-shedding.

Even for this other set of queries, the experimental results show a good correlation between the achieved SIC values and the correctness of the results. The severity of the overload condition is, in general, directly proportional to the error observed in the output results. This means that even though

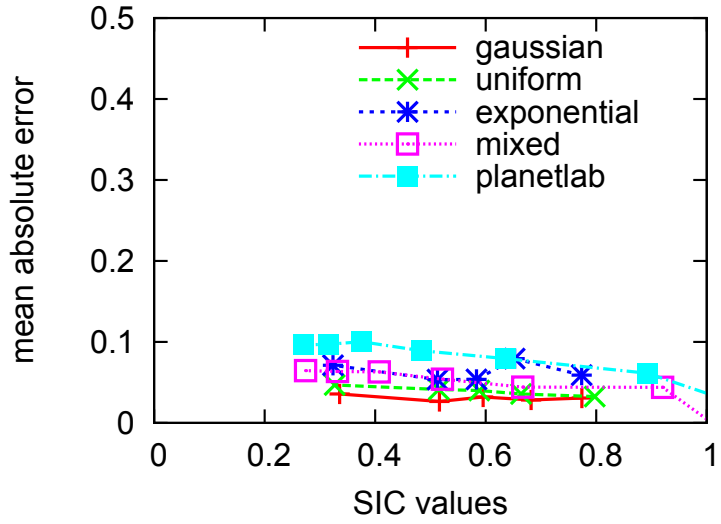

Figure 6.1: Correlation of SIC values with the query output performance for *Average* queries.

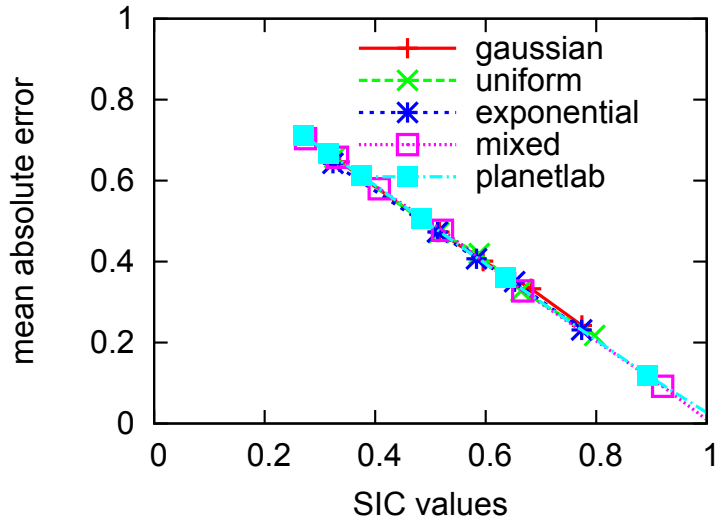

Figure 6.2: Correlation of SIC values with the query output performance for *Count* queries.

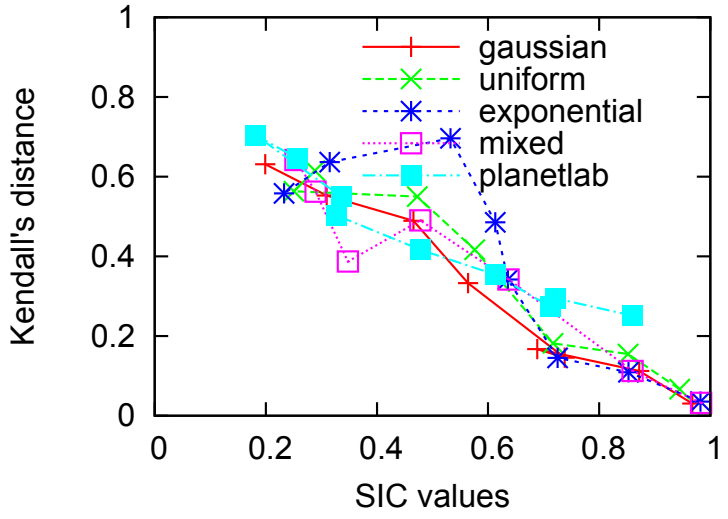

Figure 6.3: Correlation of the SIC values with the query output performance for *Top-5* queries.

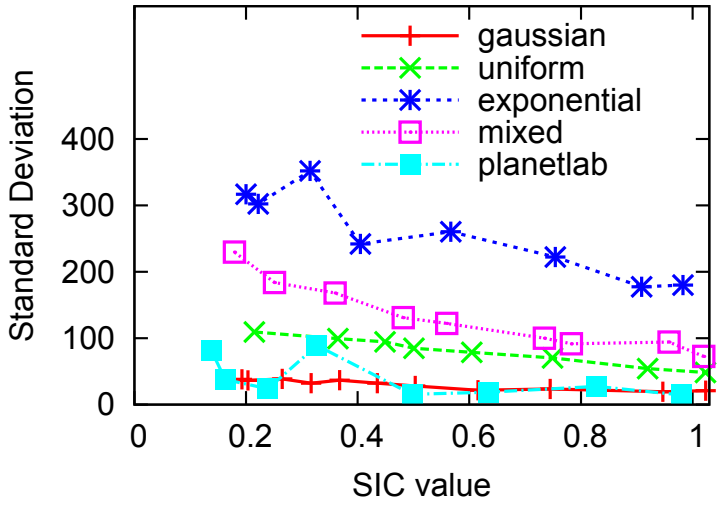

Figure 6.4: Correlation of the SIC values with the query output performance for *Covariance* queries.

the SIC metric is designed as a generic quality metric, it represents a good indicator of the quality of the computed results.

6.3 Load-shedding Policies

This section presents experiments that evaluate the use of the SIC quality metric in the context of load-shedding. Using the SIC metric, it is possible to implement *semantic shedding policies* [Ma+08] that discard tuples based on their information content. These experiments evaluate a *fair shedding* policy (see Section 5.3) that is designed to provide an equal allocation of system resources, with the goal of equalising the quality-of-service (i.e. same SIC value) for all running queries under constant overload. We compare the *fair shedding* to a *random shedding* policy.

Fairness Comparison

This section compares the random and the fair load-shedding policies. The fair shedder selects the tuples to be discarded, trying to equalise the processing degradation of all queries so that their normalised (i.e. in the [0,1] interval) result SIC values are numerically close. The random shedder, instead, picks the tuples to be discarded at random. The comparison shows that the fair shedder always outperforms the random shedder, achieving a higher average quality of the results (mean) and also a lower "spread", measured using some dispersion metrics (IQR, Q.95-Q.05 and STD). The experimental setup for this set of experiments consists of 25 Emulab nodes, as described in Table 6.1. The query workload is a mix of 3 different queries: Average, Covariance and Top-5, as described in Table 6.2.

Each run of the experiment compares the performance of the random and fair shedding policies, varying the number of query partitions (i.e. subqueries) for each query, while trying to maintain a similar total number of queries. This means that, when the the number of partitions per query increases, the total number of queries decreases (as shown in Table 6.4). Using a larger number of subqueries means that the the load of each individual query is distributed over a larger number of nodes, increasing the overhead due to the inter-node network communication. The goal is to explore the effect of increasing the number of partitions for the same number of queries, while comparing the two load-shedding policies.

Table 6.4 shows a breakdown of the load characteristics for each experimental run. The first column shows the number of subqueries that each query has been partitioned into, the second column shows the total number of deployed queries, the third shows the number of query types used (always 3), and the final column shows the total number of subqueries deployed. The last row contains the data for the *mixed* run, in which each query has been divided into a random number of partitions between 2 and 6.

Statistical Measures

The following statistical measures are used to compare the two load-shedding policies. The first, mean, is used to evaluate the average performance of the system among all queries, while the other three capture the dispersion of results. Before discussing the experimental results, we provide definitions for each of them:

Mean: The arithmetic mean is defined as the value obtained by summing all elements of the sample and dividing this value by the total number of elements in the sample. It is used to provide an indication of the central tendency of the data set.

Standard deviation: The standard deviation [Rek69] of a data set is defined as the square root of

Partitions	Queries	Total Partitions
2	334	2004
3	220	1980
4	170	2040
5	134	2010
6	112	2016
2-6	190	∼2280

Table 6.4: Workload breakdown for experiments comparing the random and fair load-shedding.

its variance. Variance is defined as the sum of the squared distances of each term in the sample from the mean, divided by the total number of elements in the sample. It shows how much variation or "dispersion" exists from the mean. A low standard deviation indicates that the data points tend to be close to the mean, whereas a high standard deviation indicates that the data points are spread out over a large range of values.

Interquartile range: The interquartile range (IQR) [UC96] is a measure of variability, based on dividing a data set into quartiles. Quartiles divide a rank-ordered data set into four equal parts. The values that divide each part are called the first, second and third quartiles. They are denoted by Q1, Q2, and Q3, respectively. Q1 is the middle value in the first half of the rank-ordered data set; Q2 is the median value in the set; Q3 is the middle value in the second half of the rank-ordered data set. The interquartile range is equal to Q3 minus Q1.

Q0.95-Q0.05: This is a measure of variability used in a similar way to the interquartile range. The ordered data is divided into 100 equal parts, called percentiles, and Q0.95-Q0.05 captures the spread of the middle 90% of the ordered data values. It shows the spread of the majority of the data values and captures a larger set of values than the IQR.

Experimental Results

The first graph, in Figure 6.5, shows the comparison between the random and the fair shedding policies in terms of average quality of the delivered results. In all the experiments, the fair shedder outperforms the random one, achieving on average results with a higher SIC value than the random shedder. The last three graphs, in Figure 6.6, 6.7 and 6.8, show the comparison between the random and the fair shedding policies in terms of variability. For all the dispersion measures used, the fair shedder delivers a lower value compared to the random shedder. This means that the fair shedder chooses a better set of tuples to be dropped, resulting in a higher mean SIC value and a lower dispersion of SIC values.

All experiments show the impact of breaking the queries into more partitions for all the analysed measures. This is due to the higher cost of inter-node communication. A larger number of subqueries provides a better load distribution among all the processing nodes, but it also increases the total load imposed on the system. The *mean* SIC value is higher in the case of two partitions and decreases as the number of partitions increases. The *dispersion* metrics, instead, have lower values for two partitions. Their value increases when increasing the number of subqueries.

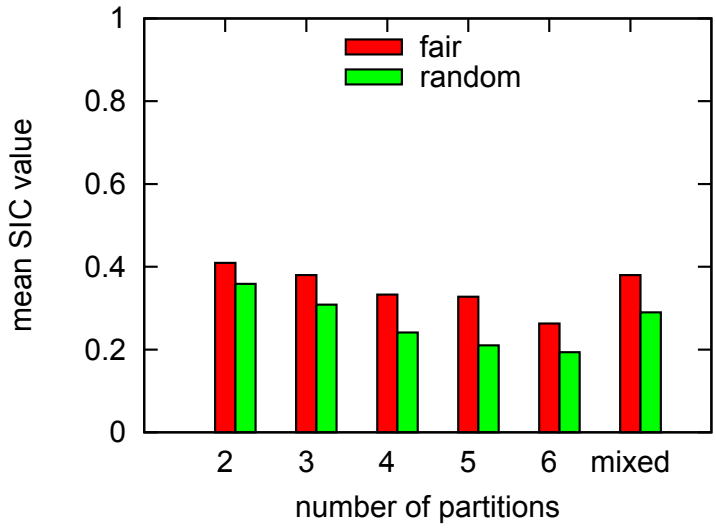

Figure 6.5: Comparison of fair and random shedders (MEAN).

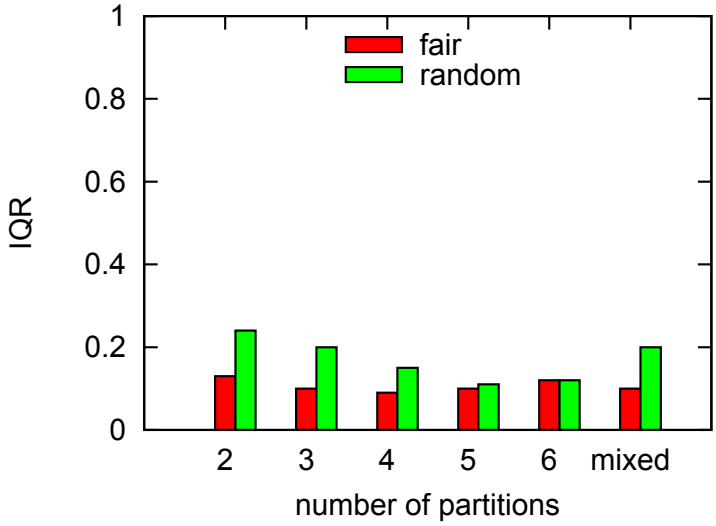

Figure 6.6: Comparison of fair and random shedders (IQR).

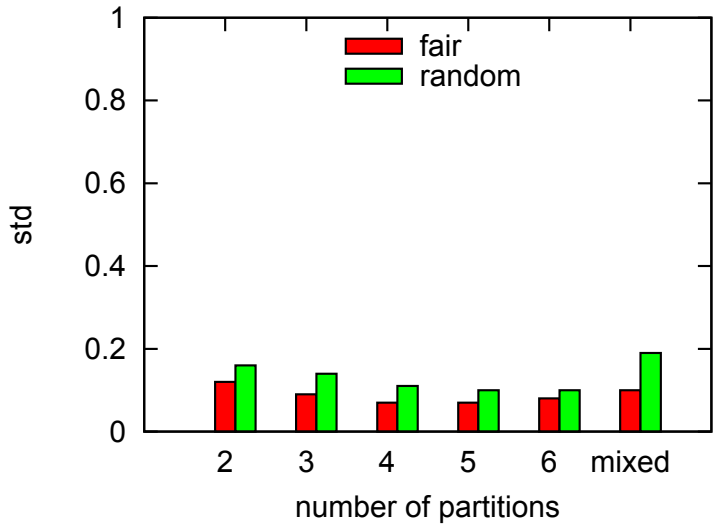

Figure 6.7: Comparison of fair and random shedders (Standard Deviation).

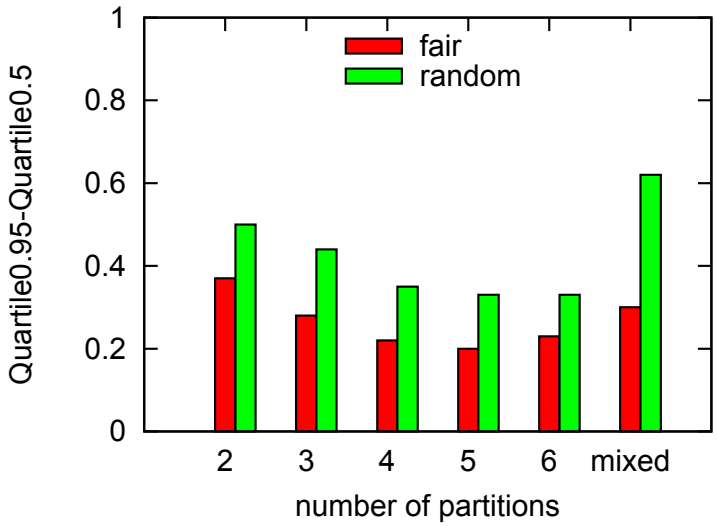

Figure 6.8: Comparison of fair and random shedders (Q0.5-Q0.95).

Fair Shedder Scalability

The following set of experiments evaluate the scalability of the *fair shedder* in terms of the number of nodes and queries. The aim is to test the variation of the SIC values of results when the amount of processing resources changes. In the first set of experiments, the amount of load remains constant, while increasing the processing resources. Adding more nodes leads to an increase in the resulting SIC values, as the amount of load-shedding required is reduced. In the second set of experiments, the variation is in terms of processing load, while maintaining a constant amount of processing resources. In this case, increasing the number of deployed queries leads to a reduction of SIC values. Both sets of experiments are deployed on the Emulab testbed, as described in Table 6.1. The processing nodes are loaded with an evenly mixed workload of COV, AVG and TOP-5 queries, as described in Table 6.2.

Increasing the Number of Nodes

This set of experiments measures the scalability of the fair shedding policy in terms of number of nodes. The aim is to measure the performance of the fair shedder when varying the amount of processing resources. Increasing the number of nodes, keeping the amount of queries stable, means progressively reducing the overload on each processing node. The fair shedder should achieve a proportionally better performance in terms of mean SIC value. Ideally, reducing the number of nodes of 50% should result in the same reduction in the SIC values of the computed results.

Figure 6.9 shows the results obtained after running the experiment on a number of nodes varying from 9 to 24. Increasing the number of processing nodes reduces the average load on each node and thus

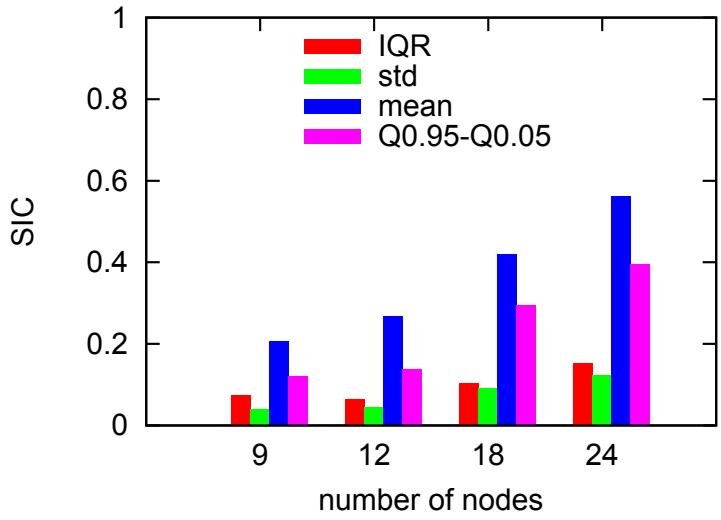

Figure 6.9: Fairness for increasing number of nodes.

the need for load-shedding. The higher amount of processing resources available leads to an increase in average SIC values of the output tuples. All dispersion measures show a reduction, which means that a better performance also is achieved in terms of the variability of the output SIC values. The value of the dispersion measures grows together with the value of the mean. If the value of the mean doubles, in fact, the value of the dispersion metrics tends to double as well, because the variability range is larger. With a mean of 0.2, we can expect the SIC values to be distributed in the interval [0-0.4], while with a mean of 0.4, we can expect the range to be [0-0.8]. Since the range is doubled also the dispersion measures should be doubled. What should remain constant is the ratio between the dispersion measure and the mean. For instance, the ratio between the standard deviation and the mean takes the name of *coefficient of variation* [Bla09]. All these ratios present a stable behaviour, confirming the good scalability of the fair shedder when varying the number of processing nodes.

Increasing the Number of Queries

This set of experiments observes the performance in terms of SIC value when varying the number of queries (i.e. increasing the load) while maintaining a constant amount of processing resources. The total number of nodes used in these experiments is 25, with the characteristics described in Table 6.1. The nodes are loaded with an increasing number of queries, with an evenly mixed workload of COV, AVG-all and TOP-5 queries (see Table 6.2).

Figure 6.10 shows the degradation in SIC values when deploying a number of queries that varies from 180 to 1200. The results show that the value of all queries is reduced proportionally to the number of queries. From the graph, it is possible to observe that the performance of the system is not strictly linear. The slope of the descending mean SIC values changes sign, meaning that at the beginning the

percentile reduction in SIC value is lower than the relative increase in number of queries. At a certain point this changes and the system performance becomes sublinear. The next section will show how it is possible to exploit this behaviour to strike a trade-off between the quality of results in terms of SIC value and the cost per query.

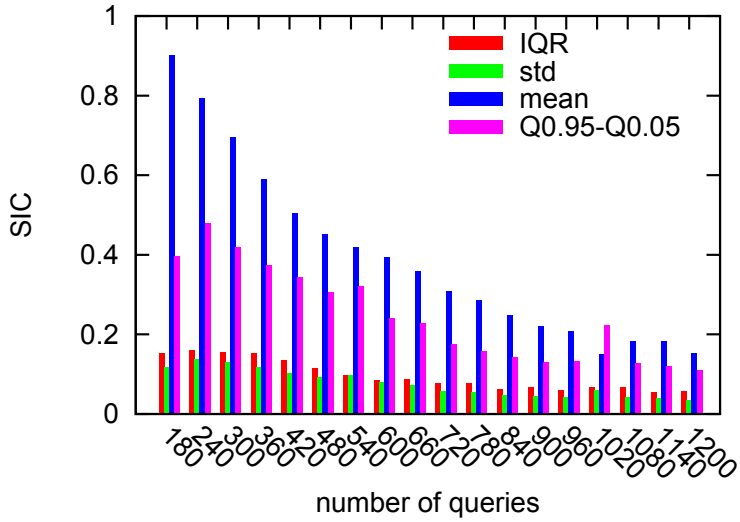

Figure 6.10: Fairness for increasing number of queries.

6.4 Cost/QoS Trade-off

When the user is willing to accept approximate results, it is possible to use the SIC metric to strike a trade-off between quality-of-service and costs. If we maintain the amount of processing resources stable, increasing the number of deployed queries leads to a reduction of the cost per query. When the percentage of cost reduction is greater than the reduction in quality-of-service (i.e. SIC values), there is a cost advantage adding more queries.

The difference in cost, $\Delta(cost)$, is calculated as:

$$\Delta(cost) = \frac{C_{base} - C_x}{C_{base}} \tag{6.3}$$

where C_{base} is the base cost for the starting deployment and C_x is the cost of deploying x queries.

The difference in quality-of-service, $\Delta(QoS)$, is calculated as:

$$\Delta(QoS) = \frac{QoS_{base} - QoS_x}{QoS_{base}} \tag{6.4}$$

where QoS_{base} is the quality-of-service value for the starting deployment and QoS_x is the quality-of-

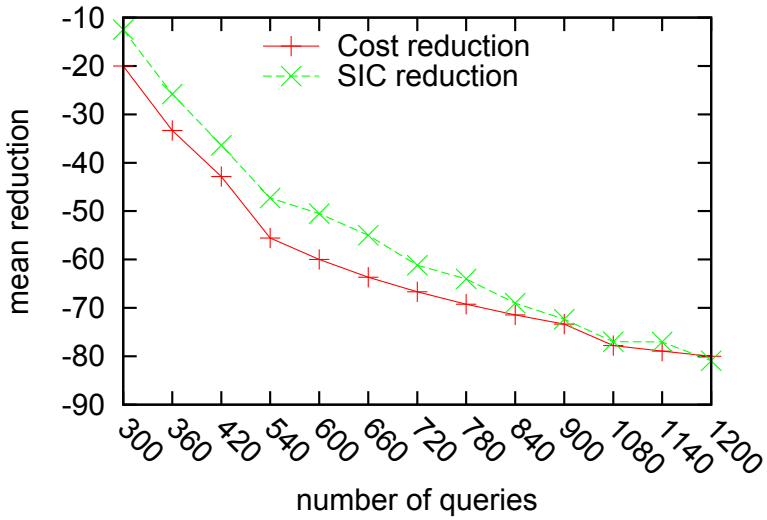

Figure 6.11: Trade-off between the reduction of cost per query and reduction in QoS expressed as mean SIC values.

service achieved deploying x queries.

The deployment of new queries is considered to be advantageous when:

$$\Delta(QoS) > \Delta(Cost) \tag{6.5}$$

The difference between the reduction in cost and quality-of-service is inversely proportional to the number of queries. The more queries are added, the lower the cost advantage, until a point is reached where increasing the number of queries is no longer advantageous. A user can keep adding new queries until this point is reached, until the difference goes under a certain threshold, or until a minimum SIC values is reached.

Experimental Results

For the trade-off evaluation, the data from Figure 6.10 is used. The cost of the infrastructure remains stable, so the cost per query is calculated as the inverse of the number of queries: $1/N_{queries}$.

Figure 6.11 uses the mean SIC value as the quality-of-service metric. The base case is at 180 queries and at every step 60 more queries are deployed. The graph shows how at 300 queries there is almost an 8% difference between the reduction in quality-of-service and the cost reduction. This difference grows smaller as more queries are deployed until a "tipping point" is reached at around 900 queries. From this point on, adding more queries is no more advantageous.

6.5 Summary

This chapter presented the experiments performed to evaluate the advantages of employing the SIC quality metric in a stream processing system. All experiments were performed using the DISSP prototype described in Chapter 4. The first set looked at the correlation between SIC values and the correctness of results. Even though the SIC quality metric was not designed as a direct measure of the accuracy of results, experiments showed that for many classes of queries there is a good correlation, confirming that a high value of the metric indicates a good confidence in the results. The second set of experiments focused on the evaluation of the *fair shedding policy*, designed to exploit the SIC metadata to evenly allocate resources among queries, comparing its performance with a random shedder. Results showed that employing the fair shedding policy leads to a more even distribution of SIC values for the delivered output tuples. Finally, a set of experiments was devised to investigate the correlation between cost and quality of the results. These experiments showed that it is possible to strike a trade-off between the achieved SIC value of results and a reduced rental cost for the processing infrastructure.