Chapter 6

Evaluation

This chapter presents 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 looks at the correlation between SIC values and the correctness of results. Even though the SIC quality metric was not designed as a direct indication of the accuracy of results, experiments show 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 focuses 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 show 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 show that it is possible to achieve a good confidence for the final results even when reducing the amount of processing resources, striking a trade-off between correctness and a sensibly reduced rental cost for the processing infrastructure.

6.1 SIC values and Correctness of Results

This section evaluates the correlation between SIC values of the output tuples and the correctness of the delivered results, across a representative range of *aggregate* and *top-k* classes of queries. These query types are used in a variety of data processing applications, such as sensor networks, social media analysis, etc. The experiments used real load measurements collected on PlanetLab and also some synthetic workloads containing inputs arranged according to some well-known statistical distributions.

6.1.1 Experimental Setup

For the this set of experiments a Local test-bed is used, as described in Table 6.1. A single DISSP processing node is overloaded by instantiating an increasing number of queries of a single type. 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 type and across five different sets of source data.

The queries chosen for this experiment belong to two main classes: aggregate (i.e. AVG, MAX and COUNT) and complex (i.e. TOP-5, AVG-all and COV1) queries; they are summarised in Table 6.2. The queries use a diverse set of operators, namely: average, max, top-k, group-by, filter, join, covariance, time-window, remote-sender, remote-receiver and output.

The input data generated by the sources contains either real-world load measurements or a specific synthetic workload. The real-world data streams are traces from CPU load and memory usage measurements from all PlanetLab nodes [wl] collected during April 2010, as recorded by the CoTop project [CoD10]. The values of the synthetic data follow either a *gaussian*, *uniform* or *exponential* distribution that have a mean of 50. Furthermore, an *advanced* synthetic workload is used, that mixes all three distributions randomly. 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.

Local Testbed	
Physical layout	3 machines of 1.8 Ghz CPU and 4 GB of memory running Ubuntu Linux 2.6.27-17-server and connected over 1 Gbps network.
DISSP layout	(1 machine: 1 oracle node), (1 machine: source data generation and query submission), (1 machine: 1 DISSP node).
Sources	400 tuples/sec in 5 batches/sec of 80 tuples/batch.
Emulab Testbed	
Physical layout	25 pc3000-type machines connected over a 1 Gbps LAN network. Each machine has a 3 Ghz CPU and 2 GB of memory and runs the FBSD410+RHL90-STD Emulab-configured Linux image.
DISSP Layout	(1 machine: 1 oracle node), (3 machines: source data generation), (3 machines: query submission), (18 machines: 18 DISSP nodes).
Sources	150 tuples/sec in 3 batches/sec of 50 tuples/batch.

Table 6.1: Testbed for correctness experiments.

Aggregate Queries	
AVG	Calculates the average value over 1 sec.
MAX	Calculates the maximum 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.
AVG-all	Shows the avg CPU consumption over PlanetLab nodes over 1 sec.
COV	Shows the covariance of the CPU consumption between two PlanetLab nodes.

Table 6.2: Query workload for correctness experiments.

The AVG, COUNT and MAX aggregate queries connect to one source with a rate of 400 tuples/sec and COV queries connect to two sources of 400 tuples/sec. The TOP-5 query connects to 20 sources, equally split between CPU load and memory readings. Each source has a rate of 20 tuples/sec. [**Check?**] To create a skewed distribution for the TOP-5 query, the mean for 10 sources is increased from 50 to 230 in steps of 20.

6.1.2 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 type, the experiment is repeated for all the different input data sets.

For the AVG, COUNT and MAX aggregate queries we use the *mean absolute error* that quantifies the relative distance of the *degraded* from the *perfect* result value across all measurements n for the duration of the experiment:

$$\frac{\sum|\frac{degraded - perfect}{perfect}|}{n} \tag{6.1}$$

For the TOP-5 query we calculate the error using the Kendall's 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. This value is normalised to lie within the [0,1] interval.

[**Check?**] In the case of the COV we use the following methodology. Each experiment produces a series of query results; the experiments duration is 5 minutes each and the COV query outputs a new *sample* covariance every 1 sec. These values are random and their expected value matched the *real* covariance. In our case, the real covariance come from perfect processing. Hence, by comparing the two variables, —i.e. the degraded sample covariances and the perfect real covariances— we can evaluate the correlation between the SIC values and the COV query results. We compare the two variables through their standard deviations.

6.1.3 Experimental Results

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

Aggregate Queries

For the AVG queries (Figure 6.1), the graph shows a small error even under heavy overload. This is due to the particular nature of the the average operation. Since the load-shedder discards tuples at random, the distribution of the data is not significantly modified. Thus, even when the resulting SIC value is very low, the computed average results are numerically similar to the ones produced in perfect conditions.

For the COUNT query (Figure 6.2), however, the opposite is true. In this case the error grows linearly with the amount of discarded data. This is an extreme case, where the occurrence of load-shedding has always a major impact on the final results. Each tuple that is discarded, in fact, reduces the total output value (final count) and increases the error.

In the case of the MAX query (Figure 6.3), we note the result has a small error for the synthetic distributions. However, the error increases and shows a linear correlation in the case of the mixed and the PlanetLab distributions. [**Check?**] Overall, the more source tuples are used for a result stream, the more accurate the result output is.

[**Note**] what else?

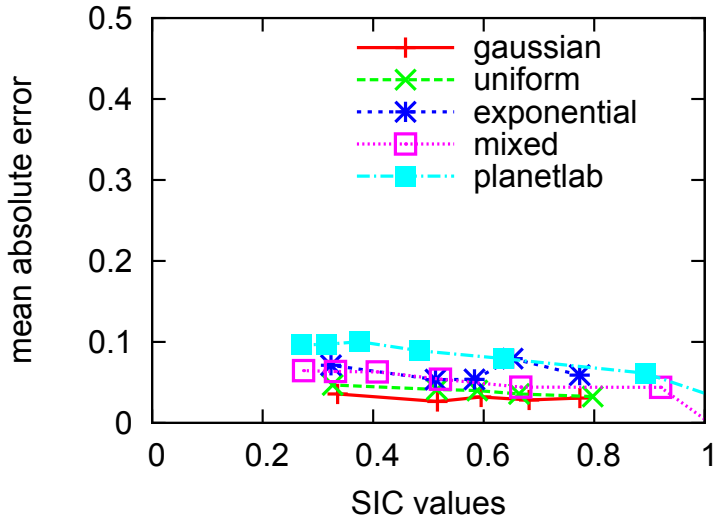

Figure 6.1: Aggregate *Average* queries, correlation of SIC values with the query output performance.

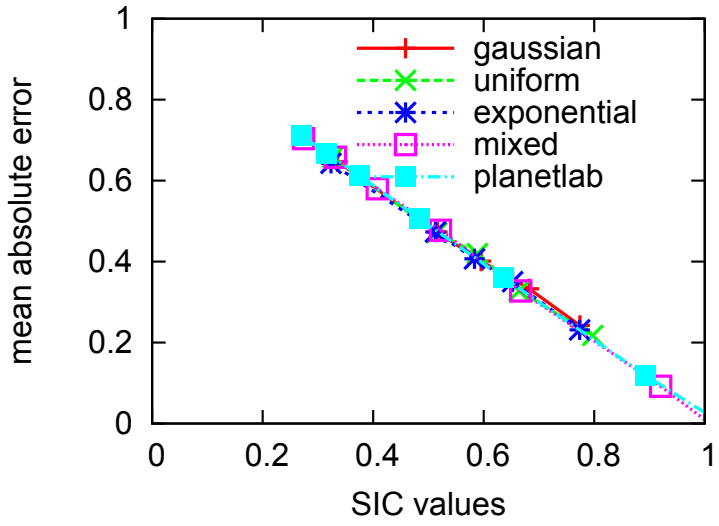

Figure 6.2: Aggregate *Count* queries, correlation of SIC values with the query output performance.

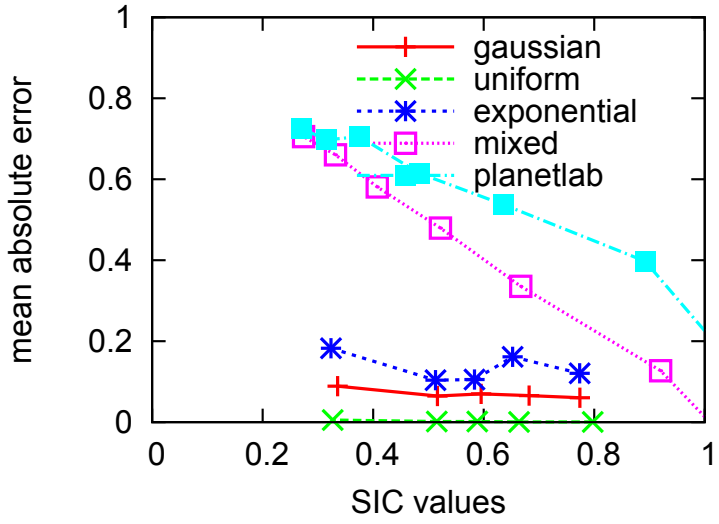

Figure 6.3: Aggregate *Max* queries, correlation of SIC values with the query output performance.

Complex Queries

For the TOP-5 query (Figure 6.4), the error expressed as Kendal distance decreases almost linearly with the amount of the performed load-shedding. [**Note**] what else to say?

[**Check?**] For the COV query (Figure 6.5), the standard deviation of the results decreases and approaches the perfect result, i.e. SIC $\rightarrow$ 1, as less source data are shed and the SIC values increase.

[**Note**] what else?

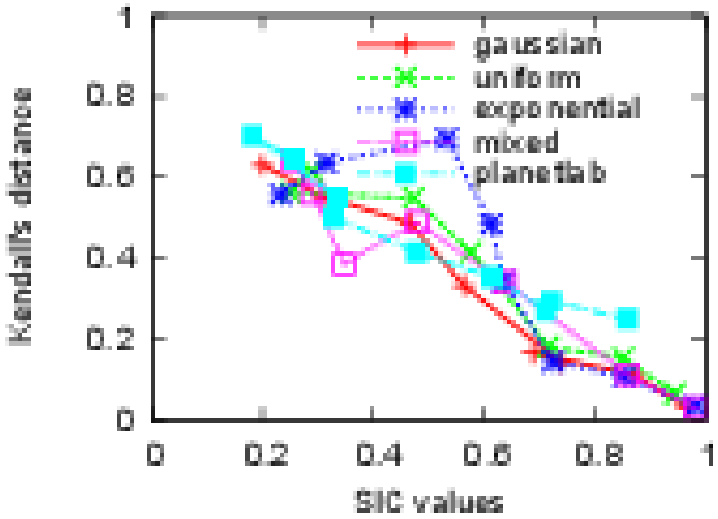

Figure 6.4: Correlation of the SIC values with the query output performance, *top-5* queries.

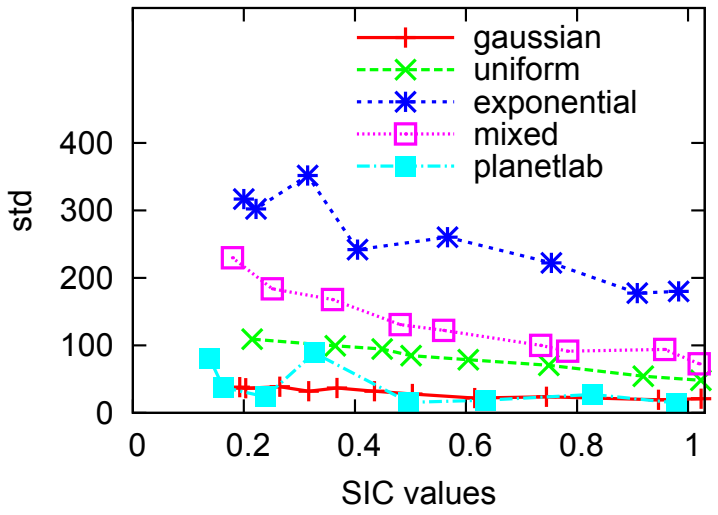

Figure 6.5: Correlation of the SIC values with the query output performance, *covariance* queries.

6.2 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* that discard tuples based on their information content. The SIC metric measures the amount of failure that occurred during the creation of a tuple and thus is an indication about the importance of the tuple towards the creation of the final result. These experiments evaluate a *fair-shedding* policy that was designed to provide an equal allocation of system resources, with the goal of equalising the quality-of-service for all running queries in a condition of constant overload.

6.2.1 Random vs Intelligent Shedding

This set of experiments has been devised to compare the *fair-shedding* and *random-shedding* policy. The fair-shedding policy is an implementation of the algorithm presented in Section 5.3.2. The goal of this policy is to achieve fairness in the resource allocation for all queries running into the system. The load-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) resulting SIC values should be numerically close. The random shedder instead, does not employ any strategy when choosing the tuples to be discarded.

6.2.1.1 Fairness Comparison

This section compares the random and the fair load shedding policies. Te comparison shows that the *fair shedder* always outperforms the *random shedder*, delivering an higher average quality of the results (mean) and also achieving a lower "spread", measured using some dispersion metrics (IQR, Q.95-Q.05 and STD).

Experimental Setup

The experimental setup for this set of experiments consisted of 18 Emulab nodes, as described in Table 6.5. The query workload was a mix of 3 different queries: Average, Covariance and Top-5, as described in Table [**?**]. Average queries calculated the average CPU and memory consumption of a chosen PlanetLab node from those present in the input traces. Covariance queries calculated the covariance of the CPU consumption between couples of PlanetLab nodes. Top-5 queries calculated the 5 PlanetLab nodes with the highest amount of available CPU and at least 100 MB of free memory.

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

Table 6.3: Workload breakdown for experiments comparing the random and fair shedding policies.

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 maintaining a constant the total number of deployed partitions. [**Check?**] This means that on each run the number of deployed queries varies, but the total load on the system remains equivalent.
[**Note**] is this correct? why is the load "constant" if the number of queries changes?

Table 6.3 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 partition between 2 and 6.

Statistical Metrics

The following statistical metrics have been 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 measure the dispersion of results. Before discussing the experimental results, let us provide the definitions for each of them:

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

Standard Deviation: The standard deviation [**?**] of a data set is described as the square root of its variance, where variance is defined as the sum of the squared distances of each term in the distribution

from the mean, divided by the number of term s in the distribution. It shows how much variation or "dispersion" exists from the average (mean, or expected value). A low standard deviation indicates that the data points tend to be very close to the mean, whereas 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; and 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: [**Check?**] The Q0.95-Q0.05 [**?**] is a measure of variability similar to the Interquartile Range. The data range is divided into 100 percentile, called percentiles, and the middle value of the 95th (Q.95) and the 5th (Q.05)percentile are used. The value of this metric is calculated as Q.95 minus Q.05.

Experimental Results

The first graph (Figure 6.6) shows the comparison between the random and the fair shedding policies in terms of average quality of the delivered results. In all the experimental setups the fair-shedder outperforms the random one, delivering on average results with an higher SIC value than the random-shedder.

The last three graphs (Figure 6.7, 6.8 and 6.9)show the comparison between the random and the fair shedding policies in terms of variability. For all the dispersion metrics used the fair-shedder delivers a lower value compared to the random-shedder.

[**Note**] what else?

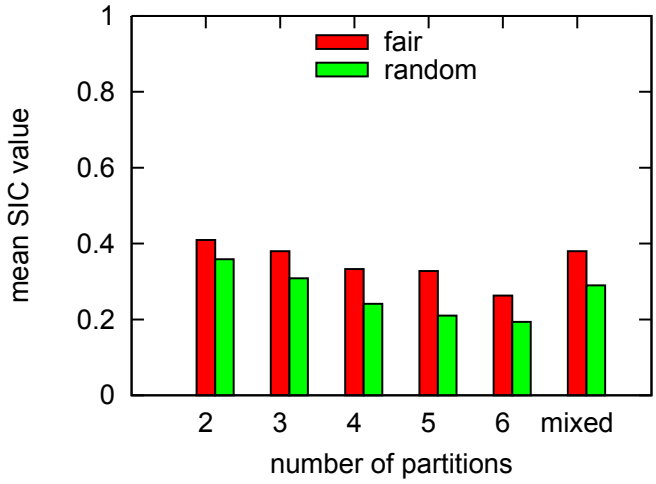

Figure 6.6: Fair and random shedders comparison: MEAN.

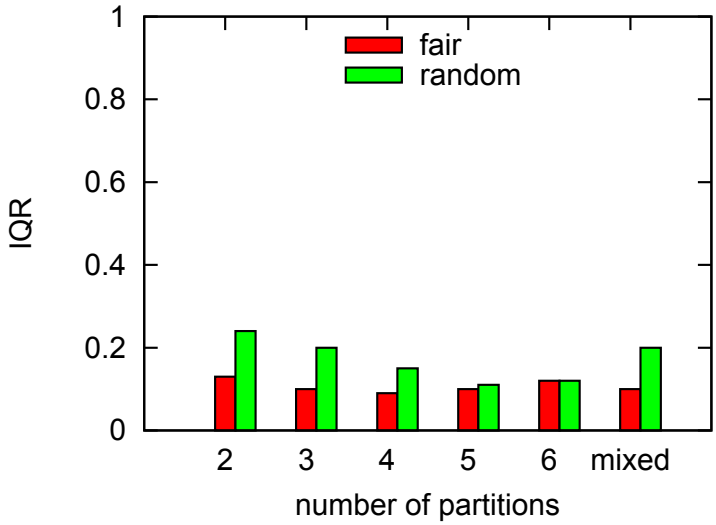

Figure 6.7: Fair and random shedders comparison: IQR.

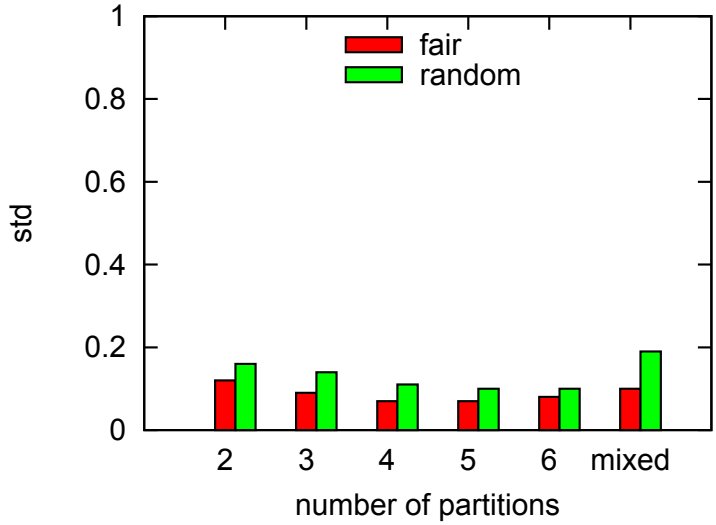

Figure 6.8: Fair and random shedders comparison: STD.

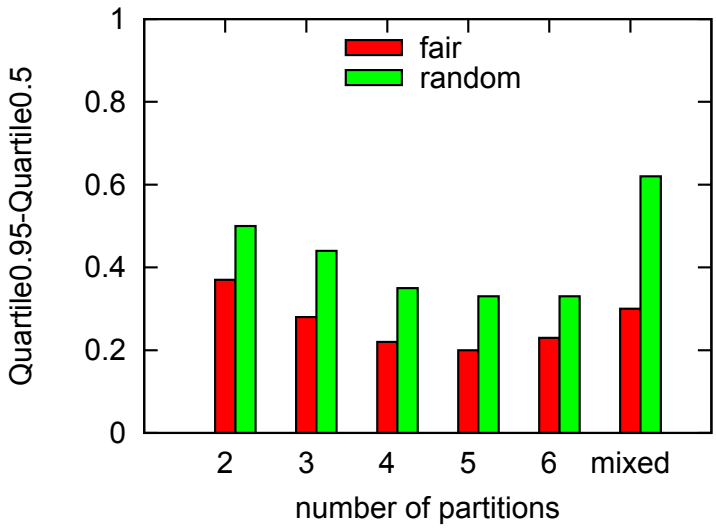

Figure 6.9: Fair and random shedders comparison: Q0.5-Q0.95.

6.2.2 Fair-Shedder Scalability

The following set of experiments has been devised to test the scalability of the *fair-shedder* in terms of increasing number of nodes and queries. The aim is to test the variation of the SIC values of the delivered results when the amount of processing resources changes. In the first set of experiments the amount of load remains constant, while increasing the amount of processing resources. Adding more computing nodes leads to an increase of the resulting SIC values, as the amount of load-shedding required is reduced. In the second set of experiments the variation is in term 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 show a proportional variation of quality-of-service, showing a good scalability of the fair-shedder.

Experimental Setup

Both sets of experiments were deployed in an Emulab testbed, as described in Table 6.5. The input data was a synthetic workload that replayed real traces of CPU load and memory usage measurements from all PlanetLab nodes [wl] in April 2010, as recorded by the CoTop project [CoD10].

The query workload was a mix of 3 different queries: Average, Covariance and Top-5, as described in Table 6.4. Average queries calculated the average CPU and memory consumption of a chosen PlanetLab node from those present in the input traces. Covariance queries calculated the covariance of the CPU consumption between couples of PlanetLab nodes. Top-5 queries calculated the 5 PlanetLab nodes with the highest amount of available CPU and at least 100 MB of free memory.

Scalability Workload	
AVG	Calculates the average value over 1 sec.
TOP-5	Shows the 5 PlanetLab nodes with the highest amount of available CPU and at least 100 MB of free memory over 1 sec.
COV	Shows the covariance of the CPU consumption between two PlanetLab nodes.

Table 6.4: Query workload for increasing number of queries experiments.

Emulab Testbed	
Physical layout	pc3000-type machines connected over a 1 Gbps LAN network. Each machine has a 3 Ghz CPU and 2 GB of memory and runs the FBSD410+RHL90-STD Emulab-configured Linux image.
DISSP Layout	(1 machine: 1 oracle node), (3 machines: source data generation), (3 machines: query submission), (18 machines: 18 DISSP nodes).
Sources	150 tuples/sec in 3 batches/sec of 50 tuples/batch.

Table 6.5: Testbed for increasing number of queries testbed.

Increasing Number of Nodes

This set of experiments was devised to measure the scalability of the fair-shedding policy in terms of number of nodes. In each iteration of the experiment, 500 queries were deployed on a fixed number of processing nodes. The nodes were loaded with an increasing number of queries, with an evenly mixed workload of Covariance, Average and Top-5 queries, as described in Table 6.4.

The aim was to measure the performance of the fair-shedder when varying the number of nodes. Increasing the number of nodes means, maintaining the query workload stable, means progressively reducing the overload on each processing node. The fair-shedder should achieve a proportionally better performance, maintaining a stable ratio between each statistical metric and the number of nodes.

6.2.2.1 Experimental Results

Figure 6.10 show the results obtained running the experiment on a number of nodes varying from 9 to 24 in four steps. Increasing the number of processing nodes reduces the average load on each node and the need for load-shedding. The lower number of discarded tuples lead to an increase of average SIC values of the output tuples. All dispersion metrics show a reduction, meaning a better performance also in terms of variability of the sample.

[**Note**] what else?

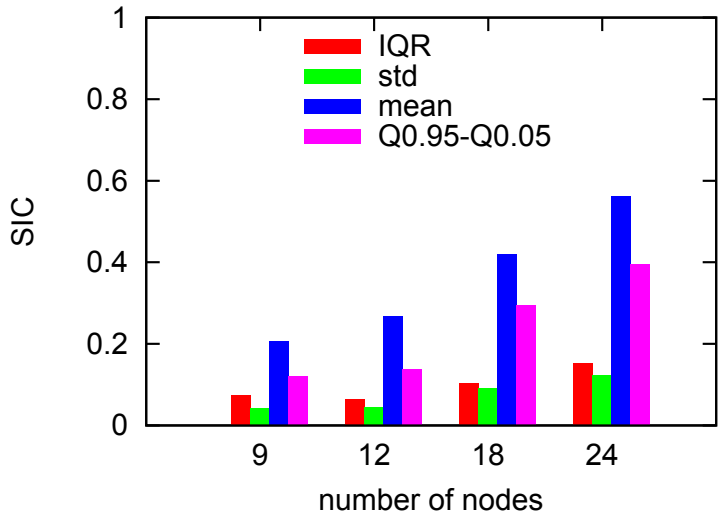

Figure 6.10: Fairness for increasing number of nodes.

Increasing Number of Queries

This set of experiments aims at observing the performance in terms of SIC value when varying the number of queries —i.e. increasing the load— while maintaing a constant amount of processing resources. The total number of nodes used in these experiments is 18, with the characteristics described in Table 6.5. The nodes were loaded with an increasing number of queries, with an evenly mixed workload of Covariance, Average and Top-5 queries, as described in Table 6.4. Each query was partition in a number of partitions between 1 and 6.

[**Check?**] This is because at the time of submission the load on each node is different and the system decides the granularity of partitioning in order to more evenly spread the load among the processing nodes.

6.2.2.2 Experimental Results

Figure 6.11 shows the degradation in SIC value performance when deploying a number of queries that varies from 180 to 1200. Results show that the value of all queries is reduced proportionally to the number of queries.

Figure 6.12 shows the *signal-to-noise* ratio, defined as the ratio between the mean SIC value and the standard deviation of the sample for all experimental runs. The results show that the system performance is fairly stable, meaning that there is graceful degradation of the quality-of-service when increasing the processing load.

[**Note**] what else?

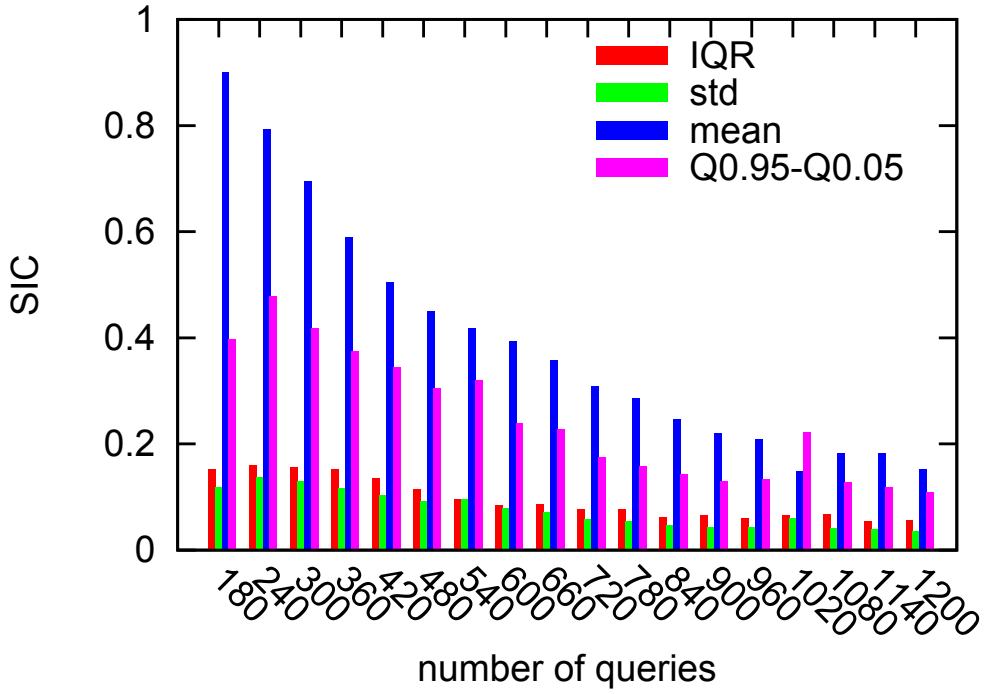

Figure 6.11: Fairness for increasing number of queries.

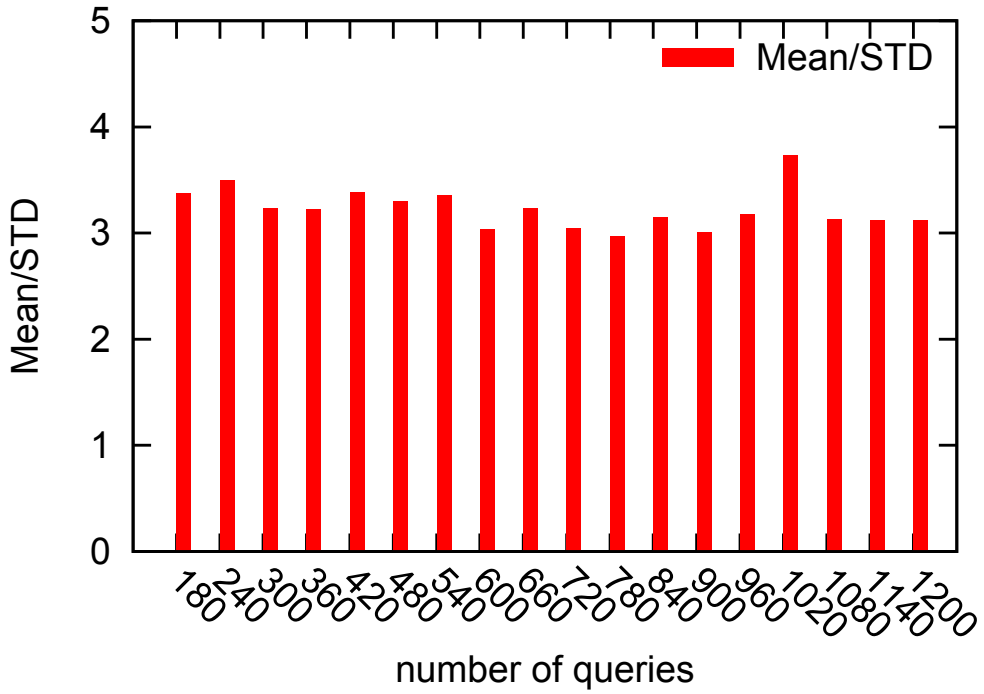

Figure 6.12: Stability of performance in terms of Mean/STD for increasing number of queries experiment.

6.3 Cost/QoS Trade-off

When the user is willing to accept approximated 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:

$$\Delta(cost) = \frac{C_{base} - C_x}{C_{base}} \tag{6.2}$$

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:

$$\Delta(QoS) = \frac{QoS_{base} - QoS_x}{QoS_{base}} \tag{6.3}$$

where QoS_{base} is the quality-of-service value for the starting deployment and QoS_x is the quality-of-service achieved deploying x queries.

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

$$\Delta(QoS) > \Delta(Cost) \tag{6.4}$$

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 6.11 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 6.13 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.

Figure 6.14 shows the same trend, but uses the Q.95-Q.05 metric as the quality-of-service indicator.

[**Note**] what else?

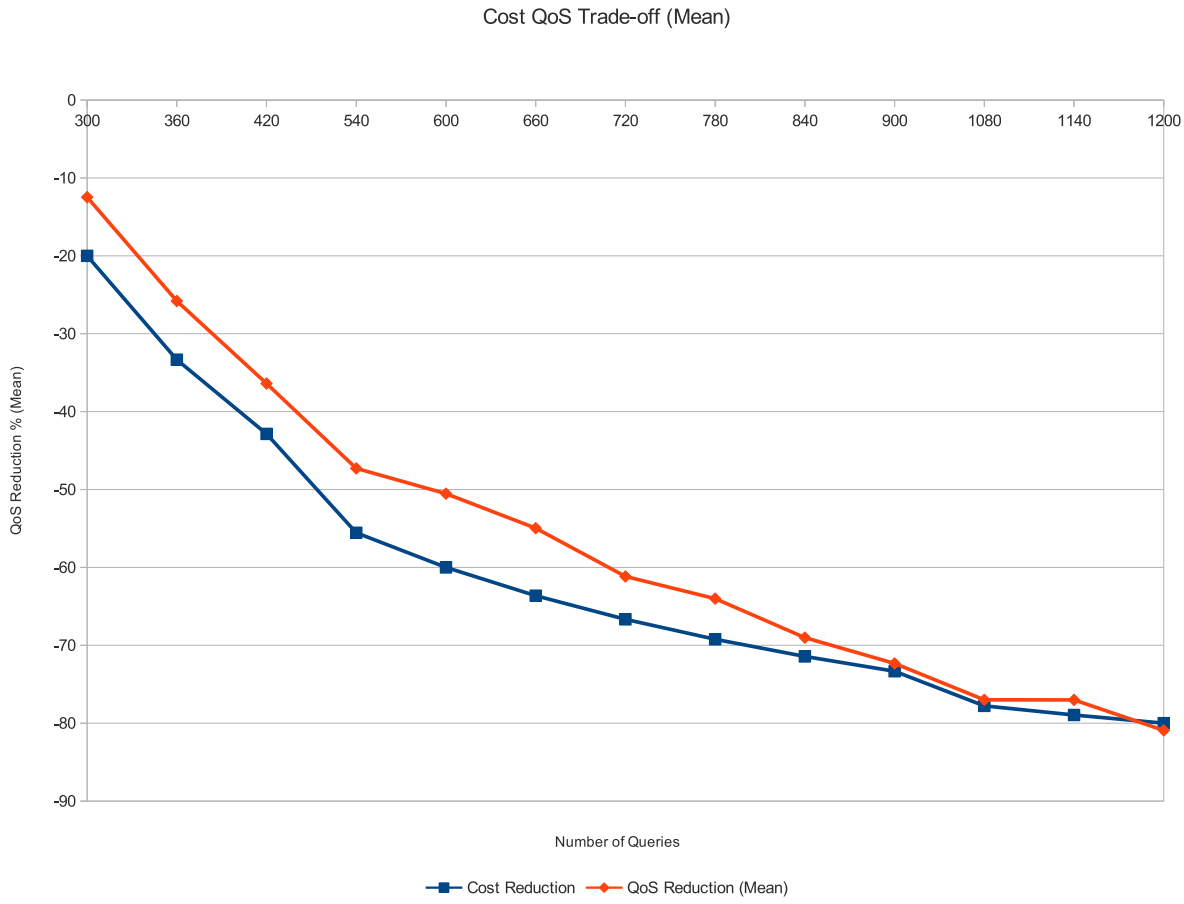

Figure 6.13: Trade-off between the reduction of cost per query and redution in Qos expressed as mean SIC values.

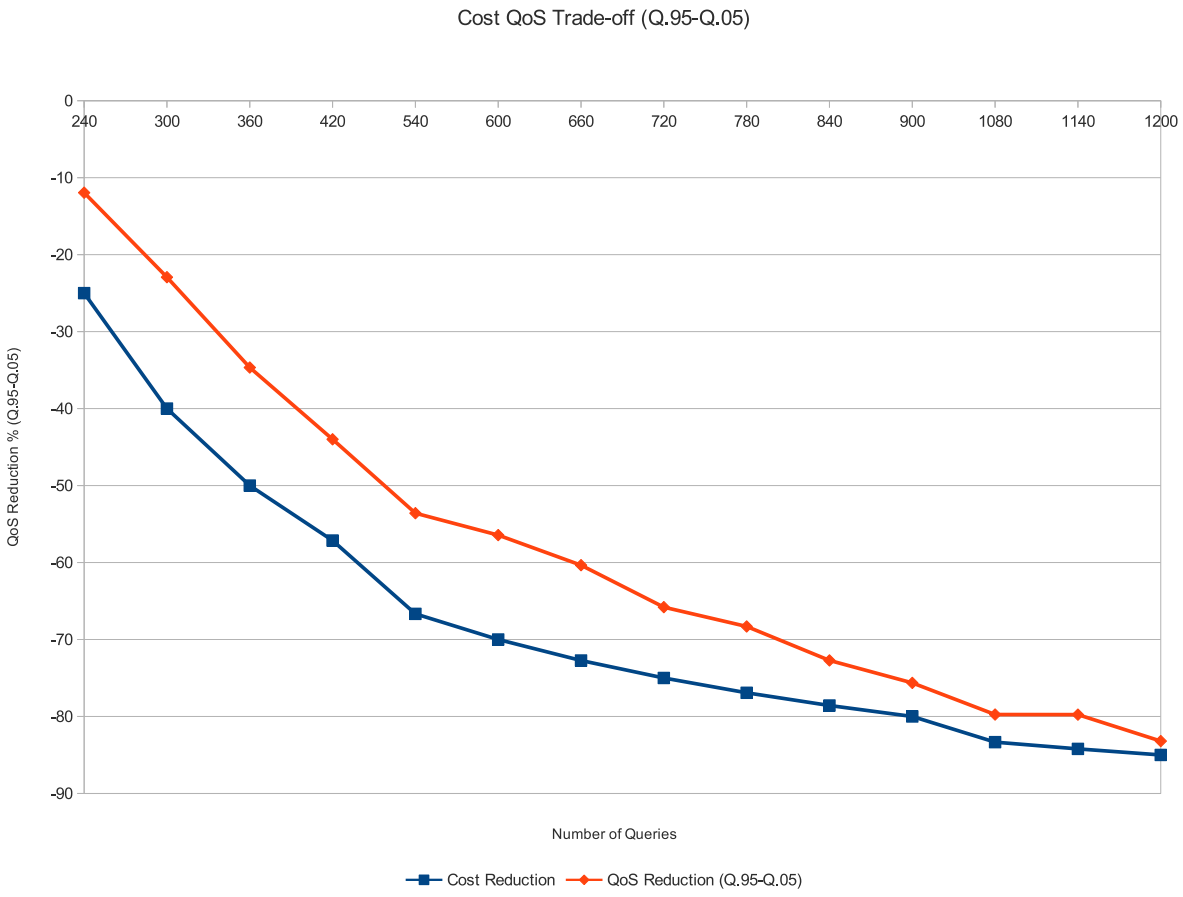

Figure 6.14: Trade-off between the reduction of cost per query and redution in Qos expressed as Q.95-Q.05 SIC values.

6.4 Source Time Window

[**Check?**] We evaluate the time-window approximation as we increase the processing and network end-to-end latency. To this end, we gradually increase the number of query fragments from one to six—thus increasing the processing delay—and we deploy each fragment in a separate Emulab node—thus increasing the network latency—while we keep the time-window fixed to 10 seconds. Overall, we perform six experiments, i.e. one for each number of query fragments, and for each we deploy 10 queries from each type of the complex workload mix. In all cases, DISSP performs perfect processing and the average result SIC is shown in Table 6.6. Results show that the STW approximation employed by DISSP correctly captures perfect processing in all cases.

[**Note**] not so sure it's clear

fragments	SIC±std	fragments	SIC±std
1	1.032±0.008	2	1.036±0.011
3	1.020±0.066	4	1.018±0.069
5	1.036±0.034	6	1.009±0.063

Table 6.6: Time-window approximation for various fragments.