Operator Placement with QoS Constraints for Distributed Stream Processing

Yuanqiang Huang, Zhongzhi Luan, Rong He, Depei Qian
Sino-German Joint Software Institute, Beijing Key Laboratory of Computer Network
Beihang University
Beijing, China
yuanqiang.huang@jsi.buaa.edu.cn

Abstract—**Distributed stream processing relies on in-network operator placement to achieve an optimal resource allocation which can use the pool of machines and network resource efficiently. Due to the QoS (Quality of Service) constraints imposed by the application, operator placement is usually treated as an optimization problem with constraints. Trying to get a global optimization is challenging since it's a NP-hard problem. In this paper, we formalize the operator placement problem with network usage as the optimization objective and use two resource allocation related QoS metrics: throughput and end-to-end delay. We propose a concept of *Optimization Power* to describe the host's capacity to reach a global optimal solution as soon as possible. We also propose a corresponding *Optimization Power*-based heuristic algorithm for operator placement. Experiment results show that our approach can achieve a better performance in terms of reducing network usage and end-top-end delay, improving success ratio, and decreasing resource discovery frequency, compared to some other placement algorithms.**

Keywords-distributed stream processing; resource allocation; operator placement; QoS;constraint;optimization

I. INTRODUCTION

Many applications require on-line analysis of large amounts of data that are being updated continuously. Such applications include sensor-based query network, financial analysis, network traffic monitoring, and so on. Some of them are called stream-processing applications, into which data tuples are pushed continuously. Complex stream-processing applications are often composed of many independent operators. Each operator has a specific functionality. Once an operator completes processing of the input, outcome may be generated, which may be related to some other operators. Operators are needed to be instantiated so that the application can be actually deployed and running. Instantiating an operator means to distribute the functionality of the operator to some physical computer, that is, a node. In practice, finding out a strong enough node to host the whole application is infeasible due to the fact that the node must meet the resource constraints of all operators and satisfy application's Quality of Service (QoS) requirements such as throughput and end-to-end delay of the data tuples. Because of the loose coupling among the operators, operators can be placed on different nodes of a network. In other words, stream-processing applications could be executed in the in-network way.

Suppose resources of the system are determinate, what we should do is to find out appropriate physical hosts in the system for operators to make the application run properly. Such a problem is called operator placement. In fact, operator placement is constrained by some factors. Firstly we must guarantee application's functional goal is reached. It means that the load introduced by the operator should not exceed the maximal capacity of the host In other words, the total resources that a host could offer should be greater than the total resource consumption of all operators running on it, otherwise the host will be overloaded, which make application work abnormally; The performance goal of the application is also very crucial. We usually use Service Level Objectives (SLOs) to denote specific performance need to achieve. So the set of hosts for an application's operators should jointly provide adequate capability to meet application's SLOs. For example, network delay between hosts may have a great influence on the final end-to-end delay from the upstream operator to the downstream operator, which may cause end-to-end delay violation when the hosts are connected by a long distance link. So the network distance among operators' hosts is also an important factor need to consider. What is more, resource costs are also an important factor since hosts and network resources are not for free in most cases. It is valuable that the application pay a minimal cost to achieve predefined functional and performance goals. It is obvious that operator placement is actually an optimization under constraints, which tries to find an optimal mapping from operators to hosts to satisfy the functional and performance goals while minimizing the cost for using resources.

This paper explores the operator placement problem in distributed stream processing. First, we formalize the operator placement as a constrained optimization problem, in which two QoS metrics: throughput and end-to-end delay, are transformed into constraints, and network usage is used for optimization objective. Some other operator placement algorithms have been proposed to solve the optimization problem with end-to-end delay as the constraint. They considered only the network delay along the path, but ignored processing delay on the hosts. We consider that the end-to-end delay consists of not only network delay but also processing delay, and take that fact into consideration when performing operator placement. To achieve this, we propose a concept of *Optimization Power* which describes the ability of the node to reduce network usage while satisfying the QoS constraints. We prove by experiments that in most cases the bigger the Optimization Power of a node is, the smaller network usage and end-to-end delay. Finally, we present a heuristic algorithm, based on node's optimization power for operator placement. Our experiments show that compared with other operator-placement algorithms, our algorithm achieves the best performance.

II. APPLICATION MODEL

Let $O = \{o_1, o_2, \ldots, o_n\}$ represent the operator set of a stream processing application. As mentioned before, an operator performs a stream processing task which continuously executes over input stream of data tuples and generates new output stream. An operator may have multiple input streams $\{is_i\}$ and produce multiple output streams$\{os_i\}$. The operators which generate output stream without receiving any input stream are called *sources*. While those which receive input streams but do not generate output are called *sinks*. The others are *intermediate operators*. Figure 1 shows a typical stream processing application in financial analysis. The financial data streams are continuously generated in real-time from the source. After the data streams go through merging, filtering and other processing at different intermediate operators, specific information is extracted and delivered to the sink eventually. Since the operators composition is normally in the form of directed acyclic graph (DAG), we use the symbol λ_{dag} to denote a stream processing application in the following statement. In this paper, we focus on the stream processing applications which have multiple sources but only one sink.

When a pair of operators having a data link in between are allocated on the same host, outputs of the upstream operator will directly be memory-copied to the input queue of the downstream operator. Otherwise data streams will be transferred between different hosts in the network. Besides the description of application's structure λ_{dag} , an operator placement request has to contain the follows: size of a data tuple $OTS^o = [ots^o_{os_1}, ots^o_{os_2}, \ldots, ots^o_{os_k}]$ and selectivity $SEL^o = [sel^o_{os_1}, sel^o_{os_2}, \ldots, sel^o_{os_k}]$ for all output streams of every operator (o), resource requirements $ER^o = [er^o_1, er^o_2, \ldots, er^o_d]$ for processing a single data tuple at every operator (o),and SLOs $Q = [q_1, q_2, \ldots, q_m]$. Selectivity $sel^o_{os_i}$ is the ratio of number of data tuples in output stream os_i to the number of data tuples in the input stream at operator o. In this paper we assume that requested application has been profiled beforehand [1], and average value for each type of meta-data has been provided. SLOs Q denotes the performance goals of application which usually include QoS metrics such as end-to-end delay, data lost ratio of data, throughput and availability, etc.

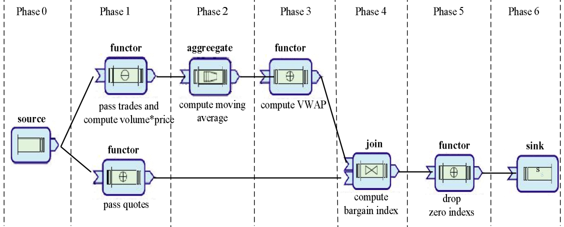

Figure 1. Application of finical analysis using distributed stream processing

III. HEURISTIC OPERATOR PLACEMENT

As we stated before, operator placement is treated as optimization problem with constraints of throughput and end-to-end delay. However it can be proved that optimal operator placement is actually a NP-hard problem, which could not get an optimal solution in polynomial time [2]. We propose a heuristic approach based on the strategy of iteratively selecting physical hosts for operators, which can mostly reduce network usage and satisfy QoS constraints.

A. Optimization Power

The first factor we need to consider while placing operators is the network delay between upstream-downstream hosts. In general, smaller network delay would lead to smaller network usage due to its definition. Network delay also has the positive contribution to final end-to-end delay.

Network delay can be reduced to zero if we place upstream-downstream operators on the same host. But it still constrained by other factors, such as node's available resource capacity. Hosts need to process a certain number of tuples per unit of time in order to achieve throughput objective, so there must be enough available resources on host to meet the specific rate requirement. Besides, host resource capacity also has the influence on application's end-to-end delay since it determines the processing delays of operators running on the host. Let $rr^{n_j}_{cpu}$ denotes residual CPU capacity on node n_j, which is the max available CPU cycles in unit of time; $er^{o_i}_{cpu}$ denotes the average CPU cycles needed by o_i to process a data tuple. Based on the approach in [2], expected time needed by an operator o_i to process a tuple on node n_j can be estimated:

$$\forall o_i, n_j \quad d_p(o_i, n_j) = \frac{er^{o_i}_{cpu}/rr^{n_j}_{cpu}}{1 - Rate^{o_i}_{in} \cdot er^{o_i}_{cpu}/rr^{n_j}_{cpu}} = \frac{er^{o_i}_{cpu}}{rr^{n_j}_{cpu} - Rate^{o_i}_{in} \cdot er^{o_i}_{cpu}} \qquad (4)$$

Taking all the factors above into consideration, we introduce an attribution named *Optimization Power* (*OP*) to measure the appropriateness of node n_k for hosting operator o, which can be calculated by the following empirical formulas:

$$OP^o_{n_k} = \left(\frac{rr^{n_k}_{cpu}}{MAX_{nd}(o, n_k)}\right)^{(1/SUM_{nu}(o,n_k))} \cdot (q^{max}_d - d_p(o, n_k) - MAX_{nd}(o, n_k)) \qquad (5)$$

$$SUM_{nu}(o, n_k) = \sum_{o_i \in US(o)} (b^{o_i}_o \cdot d_n(\Phi(o_i), n_k)) + \sum_{o_j \in DS(o)} (b^o_{o_j} \cdot d_n(\Phi(o_j), n_k))$$

$$MAX_{nd}(o, n_k) = \max_{o_i \in US(o)} (d_n(\Phi(o_i), n_k)) + \max_{o_j \in DS(o)} (d_n(\Phi(o_j), n_k))$$

Here $SUM_{nu}(o, n_k)$ and $MAX_{nd}(o, n_k)$ capture the increased network usage and maximal network delay respectively when choosing n_k to host o. $b^{o_i}_o$ is the bandwidth requirement for delivering data streams from o_i to its downstream o, $DS(o)$ and $US(o)$ denote all the downstreams and upstreams of o respectively, and $d_n(\Phi(o_i), n_k)$ denotes the expected time for transferring a byte from $\Phi(o_i)$ to n_k in which $\Phi(o_i)$ represents the node hosting operator o_i . Intuitively, the optimization power of a node for hosting an operator is considered higher if it has more residual CPU capacity which means the processing delay of the operator on this node will be smaller. And also *OP* is considered higher if the increased network usage or maximal network delay is smaller, which means the final network usage or end-to-end delay will become smaller. In addition, we consider give a node higher

OP when it produces smaller processing delay and network delay. Therefore higher *OP* will lead to smaller network usage and smaller end-to-end delay, and it actually describes the ability of a node to reduce application's overall network usage and end-to-end delay when it hosts application's operator. Node with negative *OP* value will be considered ineligible since the introduced delay is greater than maximal delay allowed (q_d^{max}). We have executed several experiments and observe that *OP*'s estimation is rather accurate in most cases.

B. *Algorithm*

Now we present an *Optimization Power-based* (OPB) operator placement algorithm. In general, our algorithm iteratively search hosts for hosting operators in order to find a locally optimal mapping between operators and hosts, which can lead to a minimal network usage and satisfy QoS constraints. The algorithm can also be triggered during runtime of application when actually generated network usage or QoS can't meet the specified requirements. To find appropriate hosts in a large node space, our algorithm relies on Resource Discovery Service (RDS) [5] to discover potential hosts that can satisfy resource requirements for processing operators. Network Coordinate Service (NCS) [6] is employed to estimate network delay between any pair of nodes using Euclidean Distance between their given network coordinates. In our case, node's information including residual resource state and network coordinate are acquired by means of RDS, and the information is used to estimate nodes' *OP* and network delays.

All intermediate operators are not assigned to hosts at the first placement, we search hosts for each operator according to the order of increasing operator's *Phase Number* (*PN*). *PN* denotes the phase of data flow in application. All sources are initialized with *PN*=0, and any other operator's *PN* is determined according to the equation: $PN(o_i) = max_{PN}(US(o_i)) + 1$, in which $max_{PN}(US(o_i))$ is the biggest *PN* in all upstreams of o_i. As show in Figure 1, each phase can have several operators, and these operators can be placed simultaneously. Even so, hosts for some operator's downstreams still can't be found at the first placement. In this case, we replace them with the sink hosts in order to calculate node's *OP*, which is not shown in the algorithm of Figure 2.

Resource discovery is seen as the most expensive action causing the most overhead in our algorithm. To reduce the overhead of resource discovery, we use the sliding query window on operators' resource requirement. Query can be for any type of resource, we choose the CPU resource because it has the strong relationship with processing delay. Since we hope to find the most appropriate node to host operators while reducing the network delay between operators and also the network usage, the algorithm always prefers to use nodes with more residual resources. ρ is one of the parameters to control resource discovery overhead. On the one hand, large ρ will lead to less frequent resource discovery; On the other hand, with increase of ρ, the size of query window becomes larger, which means the cost of a resource discovery may increase. Besides, the size of query window is also determined by the maximal resource requirement for a single host. According to the formula to calculate processing delay of operator above, we can infer the CPU requirement for host to achieve specified processing delay. As illustrated in *line* 1 in Figure 2, we calculate the maximal CPU capacity requirement for a single node to achieve specified processing delay $K_1 \cdot q_d$ which is the sum of processing delays of all intermediate operators. K_1 represents the contribution of the total processing delay to the final end-to-end delay, which is another parameter to control resource discovery overhead. Large K_1 makes the size of query window smaller to reduce the cost of a resource discovery, but takes higher risk of distributing operators over different hosts, which increase network delay and network usage. In our algorithm, ρ and K_1 take the value of [0,1]. We set ρ and K_1 to 0.2 and 0.5 respectively in our experiments. Moreover, to further reduce the resource discovery frequency, a strategy of caching query results is adopted. Since our method relies only on calculating node's *OP* and the same query window could be used for all operators, the query results can be reused. Resource discovery is triggered only if all cached nodes are not qualified.

As stated before, we want to find the host with the most optimization power to reduce network usage and to satisfy QoS constraints, the query results about node information are sorted in descending order according to their *OP* for each operator. Then we check the sorted nodes in turn to find out if some node can provide enough capacity for the operator in terms of each type of resources, and if the maximal local single-path delay from sources to the current operator ($Max(\{l_o^s | s \in S\})$) is below the specified end-to-end delay threshold. After each operator has been assigned a host, the whole procedure will be repeated until the final network usage does not increase for K_2 times. K_2 gives a tradeoff between the algorithm's overhead and the performance. In our experiments, we set K_2 to 3, which means if we find that the operator placement does not lead to a smaller network usage in three

$$TR_{cpu} = \frac{\sum_{o \in O} er_{cpu}^o}{K_1 \cdot q_d} + \sum_{o \in O} Rate_{in}^o \cdot er_{cpu}^o$$

```
loop:
    z = ⌈1/ρ⌉;
    for PN = 1 to M in λ_dag  (M is the biggest PN in λ_dag)
P1:     for every operator o in λ_dag.Phase[PN]
P2:         while (S_cache == NULL):
                if z <= 0
                    return no feasible placement solution is found;
                else
                    query = ((z − 1) · ρ · TR_cpu ≤ rr ≤ z · ρ · TR_cpu);
                    z = z − 1;
                S_cache ← invoke RDS for nodes with query;
            S_cand ← sort S_cache in descending order according to node's OP value;
            for k=1 to N in S_cand (N is the size of S_cand)
                if each R_i^o < S_cand[k].rr_i and Max({l_o^s | s ∈ S}) < q_d
                    φ{o} ← S_cand[k];
                    goto P1;
            S_cache = NULL;
            goto P2;
    if NU(λ_dag) < MIN_nu
        MIN_nu = NU(λ_dag);
        MIN_φ = φ;
    else
        count++;
    if (count >= K_2)
        return MIN_φ;
```

Figure 2. *Optimization Power*-based operator placement algorithm

times, the algorithm will stop and output the mapping of operator-host which generates the smallest network usage.

IV. EVALUATION

A. Experimental Settings

In order to evaluate the network topology influence on the OPB algorithm, we use a trace data from PlanetLab network platform [7], which includes a span of 10 months (July 2007--April 2008) collecting for network delay of every PlanetLab node pair. The total number of nodes is more than 240 and the total number of records is over 110,000. Based on these data of network delay, we can generate network coordinate for every PlanetLab node using Vivaldi algorithm [6]. Since the data of bandwidth between node pair is not provided in the trace file, we used the BRITE [4] to simulate the bandwidths. Bandwidth distribution is based on exponential model with the value range of [10KBps, 10MBps]. Besides, we adopt Random, Exponential, Normal and Zipf distribution model for resource distribution of nodes. In our experiments, we considered 3 types of important node resource: CPU speed, memory size and disk size. Each resource is assigned a value in the range of [2000, 20000].

We consider a synthetic application which consists of 10 operators including 2 sources, 1 sink and 7 intermediate operators. Sources and sink are fixed to some hosts. Moreover, every non-sink operator can have 1 to 3 downstream operators. By default, source's stream output rate is 5 tuples per second. Selectivity of all intermediate operators is set to 1.0, and the average size of tuple is 10 bytes. In the experiments, we define two adjustable factors f_{tp} and f_d to control application's throughput and end-to-end delay objectives respectively. f_{tp} is for controlling throughput objective. The stream output rate of the sources is $5 \cdot f_{tp}$ tuples per second. In same phase, half of intermediate operators set their selectivity to f_{tp}, and the other half set to $1/f_{tp}$. So when f_{tp} is getting large, throughput of the whole application is increasing and workloads of operators are differentiated in the same phase. f_d is the other factor for end-to-end delay objective. Let l denotes the maximal delay between the source hosts and the sink host. So we set the application's end-to-end delay threshold to $f_d \cdot l$, l is unchanged during simulation since the positions of sources and sink are fixed beforehand. So by adjusting f_d, we can control the overall end-to-end delay.

To make the evaluation more convincing, we also implemented three alternative operator placement algorithms for comparison: i) SBON algorithm proposed in [8] assigns optimal virtual network coordinate for every operator based on Force-Energy theory, and then perform the k-nearest neighbor search (we set k=10) for each operator in the node space to find a host which has enough resource among these k neighbors. ii) MIN-DELAY algorithm does a global search in node space for every operator to find a host which can introduce the minimal delay which is the sum of total processing delay on hosts and network delays from the current operator to the source and sink. iii) RANDOM algorithm assigns a random host for every operator. For all the algorithms, when no eligible node which can meet application's SLOs is found, placement fails.

B. Results and Analysis

We compared four algorithms above under different resource distribution: Random, Exponential, Normal and Zipf. Since the results are very similar, we only give the comparison results under Zipf distribution in the following figures below due to space limit.

Network Usage and End-to-End delay

First of all, we show the network usage and end-to-end delay achieved after one-time operator placement by these

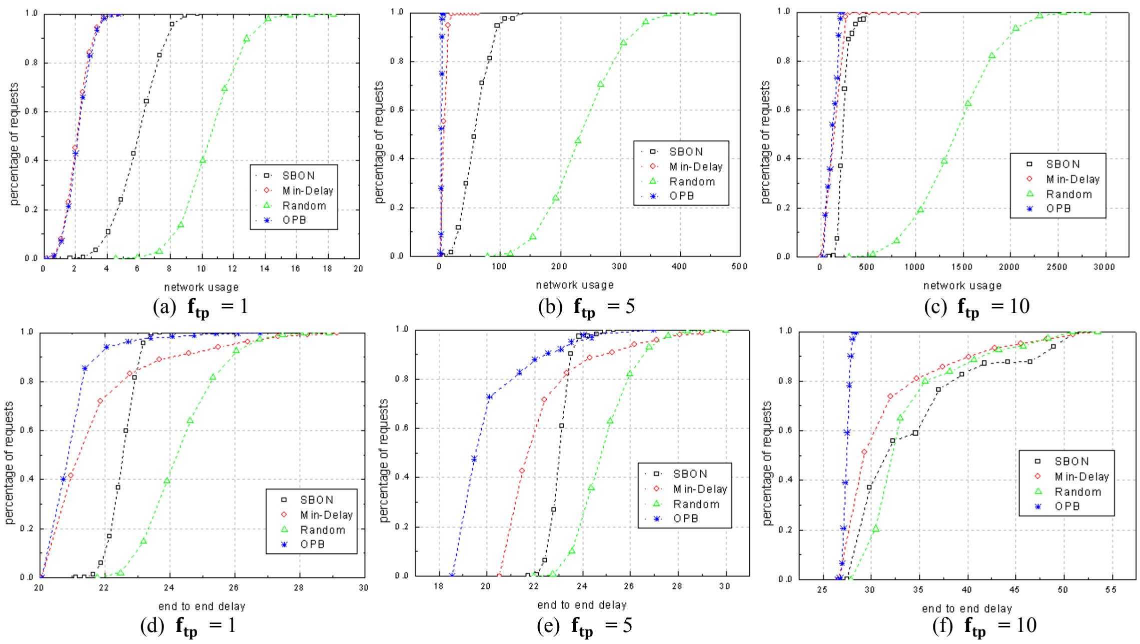

Figure 3. Comparision of Cumulative Percentage Distribution of 5000 placements for network usage and end-to-end delay with different value of f_{tp}

four algorithms. No constraint is declared on the end-to-end delay objective since we want to know the algorithms' optimization power on this QoS. From Figure 3(a), we can see OPB and MIN-DELAY algorithm generate almost the same better network usage compared with other two when f_{tp} is set to 1 which means throughput is minimal and operators are homogenous. In details, 98.5% of the 5000 operator placements using OPB or MIN-DELAY achieve network usage less than 3.89, but only about 7% achieve the same network usage level when using SBON. The RANDOM algorithm gives no network usage less than 4.0. Figure 3(d) shows the similar conclusion about the end-to-end delay that OPB is the best with 95% of placements achieving a delay less than 22, MIN-DELAY is the second best with about 73%, SBON and RANDOM are after MIN-DELAY with about 5% and 1% respectively. With the increasing of f_{tp}, the network usage and end-to-end delay of all placements using different algorithm increase simultaneously as illustrated in Figure 3. It is because that the increase of throughput makes the bandwidth consumption and host load increase. The slope change of the distribution of network usage and end-to-end delay shows that each algorithm has different sensitivity to varying throughput. From Figure 3, we can learn that variation of throughput always has the minimal impact to OPB, which means OPB is the least sensitive to the change of throughput and performs the best in reducing network usage and end-to-end delay. Different from the result of comparing network usage, SBON has the worst performance in reducing end-to-end delay when throughput is large. We think it is due to the fact that SBON use only network delay between hosts as the main contribution for the end-to-end delay. But in reality, processing delay of the operator can also affect the overall delay significantly when workload on the operator increases.

To further verify our observation on algorithms' sensitivity to varying throughput, we conduct a set of experiments on random-generated application scenarios. Table I gives the statistical data about network usage and the end-to-end delay with f_{tp} from 1 to 10 respectively. By calculating data variance for all 4 algorithms, we get the results that the variance of network usage and end-to-end delay for SBON are 9853.31and 34.35, for MIN-DELAY 367.73 and 6.06, for RANDOM 202058.02 and 18.64, and for OPB 364.9 and 4.54. So we conclude that OPB< MIN-DELAY<SBON<RANDOM in sensitivity for network usage and OPB<MIN-DELAY<RANDOM<SBON in sensitivity for end-to-end delay, which is consistent with the previous experiments.

TABLE I. COMPARISION OF NETWORK USAGE AND END-TO-END DELAY WITH DIFFERENT VALUE OF f_{tp}

f_{tp}	SBON		Min-Delay		Random		OPB	
	NU[a]	*ED*[b]	*NU*	*ED*	*NU*	*ED*	*NU*	*ED*
1	7.56	22.95	3.94	22.44	13.015	23.73	3.95	21.06
2	16.83	22.99	5.97	22.48	27.709	26.29	4.02	22.21
3	29.77	23.09	8.01	22.57	67.417	23.01	4.05	22.29
4	46.39	23.24	10.1	22.72	168.31	27.04	4.08	22.45
5	82.49	23.24	12.1	22.98	158.71	26.19	4.11	22.71
6	104.8	23.59	14.1	23.38	406.13	25.41	4.14	23.09
7	227.1	23.88	16.1	23.98	378.66	27.91	3.95	23.71
8	204.7	24.52	18.2	24.96	478.41	26.14	10.75	24.69
9	238.9	27.26	20.2	26.67	1319.1	28.15	22.62	26.36
10	252.1	41.94	70.4	30.10	1076.2	38.63	64.26	28.04

a. network usage

b. end-to-end delay

Success Rate

We think success rate is another indication of performance for operator placement. Here success rate is defined as the rate of successful placement that meets application's SLOs to the total number of placement requests. We consider throughput and end-to-end delay as QoS metrics in the experiments. As illustrated before, factor f_d is used to control the end-to-end delay objective. A large f_d results in a large end-to-end delay threshold, and relaxes the corresponding constraint to the increase of the success rate. Figure 4(a) shows the change of success rate of total 1000 placement requests with different f_d. First, we fix f_{tp} to 5 so that the throughput objective has the same effect on success rate in all experiments. We can see from the figure that OPB always achieve the highest success rate, and the success rate increase continuously with f_d increasing. The success rates of other three algorithms fluctuate at some points when f_d increasing. Similarly, to make end-to-end delay objective has the same effect on success ratio during all experiments, we fix f_d to 10. Since we fix f_{tp} to 10 so that the end-to-end delay objective has the same effect on success rate in all experiment settings. Since f_{tp} corresponds to throughput, large f_{tp} leads to increase of resource requirement of some operators, which reduces the space of eligible nodes for hosting operators. So success rate would be low if f_{tp} is small. Figure 4(b) shows the change of the success rate of total 1000 placement requests with f_{tp} increasing. We can also see that OPB still achieves the highest success rate, and the success rate decreases continuously with f_{tp} increasing.

Resource Discovery Frequency

For SBON, MIN-DELAY and OPB algorithms, the main runtime overhead is resource discovery for querying eligible potential hosts. In our study, we assume that they rely on the same resource discovery service and have the same overhead for a discovery. So by counting times of resource discovery, we can roughly compare their overhead. The RANDOM algorithm is not considered, because it performs worst in the other aspects of performance. We counted the average times of resource discovery for all successful one-time placements. From Figure 5 we can see that the number of resource discovery of SBON and MIN-DELAY is not affected by variation of the throughput or end-to-end delay objective. This is because that they have to do a resource discovery for each operator, and the discovery results can't be cached and reused. Since our application has 7 intermediate operators for placing, the number of resource discovery is always 7. Benefiting from the result caching, OPB introduces less resource discovery in all experiments although the frequency of resource discovery changes with varying SLOs. When f_{tp} is increasing, some operators have more resource requirements for hosts, which reduce the probability of finding eligible hosts in one resource discovery and requires more resource discovery to find hosts. When f_d is increasing, we found discovery overhead increases at the beginning, which is explained by the fact that the probability of finding eligible hosts in one resource discovery is dominated by the resource distribution at that time. With more and more operators are placed, the number of eligible candidates for new operator decreases, resulting in more discovery. But after some turning point, the end-to-end delay constraint becomes less significant so that the number of eligible candidates increases again. Since ρ, the parameter controlling resource query in OPB, is set to 0.2 in our experiments, the number of resource discovery in one-time operator placement will not be greater than 5.

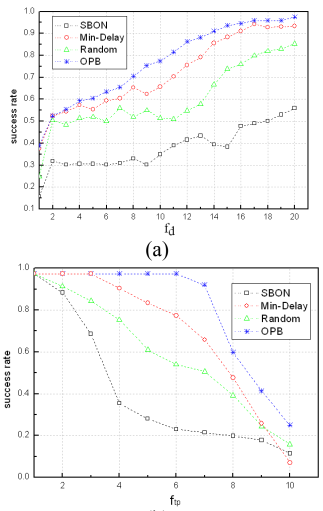

Figure 4. Comparison of success ratio of 1000 placement requests

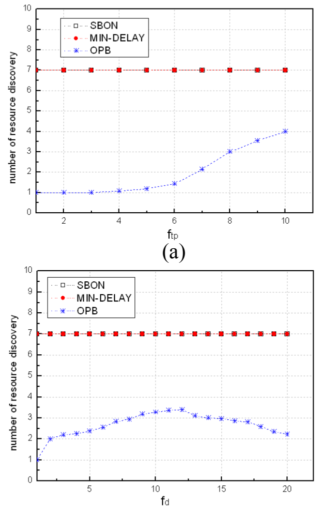

Figure 5. Comparison of resource discovery frquency

V. RELATED WORK

Many different operator placement algorithms have been proposed in the literature. As a common optimization objective for stream processing, network usage was discussed in previous studies [9]**.** All these researches make an effort to minimize the network utilization in order to reduce cost of transferring data stream over the network. [8][10] tries to solve an unconstrained optimization problem that strives for minimizing network usage using the continuous latency space, and achieves a good approximation of the unconstrained optimum.

However, minimal network usage without constraints is not always acceptable due to the QoS requirements from applications. End-to-end delay (latency) is the most discussed QoS metric in the literature. In general, given an upper bound for network delay, an optimal operator placement is required to ensure that the entire estimated network delay after placement would not exceed the given threshold [11][12]. [13] takes throughput as a QoS metric if the users of the stream-processing system provide operator-specific resource requirements as input to the placement algorithm. To meet a given throughput objective, operator placement also needs to consider the adaptation of machine and bandwidth resources. Component composition algorithm is a variant of operator placement algorithm when operators are bounded to specific hosts. In those algorithms, optimal placement is selected from all possible ones starting from the leaves and combining the candidate solutions in a bottom-up fashion [14]. On the contrary, our proposed algorithm does not introduce large overhead for finding all possible placements. Actually we don't need to check exhaustively all the nodes, because we only want to reach a local optimum with reasonable overhead, meanwhile achieving relatively high success rate. Following a similar procedure as the approach of Gu et al., Synergy in [2] calculates all possible placements and check QoS constraints for the new as well as the already deployed applications by using impact projection. Although it's a challenging problem for operator reuse, in this paper we focus on the independent operator placement and continuous adaptation during runtime.

VI. CONCLUSION

By using network usage as the optimization objective, and SLOs as constraints, we formalize operator placement as a constrained optimization problem. Since no global optimal solution can be obtained in polynomial time, we rely on heuristic approach to get a local optimal solution. To make the local optimal solution closer to the global one, we propose a *Optimization Power*-based operator placement algorithm (OPB), in which the *Optimization Power* describes node's capacity to reduce network usage and to satisfy QoS constraints. Our experimental results show that OPB has performance advantage compared to some other operator placement algorithms. We leave this problem to be explored in our future work.

ACKNOWLEDGMENT

This work is supported by NSF of China (90812001), 863 Program of China (2010AA012404), and International Cooperation Project of MOST (2009DFA12110).

REFERENCES

[1] T. Abdelzaher, "An automated profiling subsystem for QoS-aware services," in Proceedings of 6th IEEE RTAS, Washington, DC, June 2000.

[2] M. R. Garey and D. S. Johnson, "Computers and Intractability, a Guide to the Theory of NP-Completeness," W.H. Freeman and Company, 1979.

[3] T. Repantis, X. Gu, and V. Kalogeraki, "QoS-aware shared component composition for distributed stream processing systems," IEEE Transactions on Parallel and Distributed Systems (TPDS) 20, 7 (July 2009), pp. 968–982.

[4] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers, "BRITE: An Approach to Universal Topology Generation," in Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, Cincinnati, Ohio, August 2001.

[5] C. Schmidt and M. Parashar, "Flexible Information Discovery in Decentralized Distributed Systems," in Proc. of HPDC, 2003, pp. 226−235.

[6] Dabek F, Cox R, Kaashoek F, Morris R, "Vivaldi: A decentralized network coordinate system," SIGCOMM Computer Communication Review, 2004,34(4):15−26.

[7] PlanetLab Trace Data Set. http *colon slash slash* ridge *dot* cs *dot* umn *dot* edu *slash* pltraces *dot* html.

[8] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in Proceedings of 22nd ICDE, Atlanta, GA, USA, April 2006.

[9] B.J. Bonfils and P. Bonnet, "Adaptive and Decentralized Operator Placement for In-Network Query Processing, " in Proc. Conf. Information Processing in Sensor Networks, Springer-Verlag, 2003, pp. 47–62.

[10] S. Rizou, F. Durr, and K. Rothermel, "Solving the Multi-operator Placement Problem in Large Scale Operator Networks," in ICCCN'10: Proceedings of the 19th IEEE International Conference on Computer Communication Networks . 2010.

[11] V. Pandit et al, "Performance Modeling and Placement of Transforms for Stateful Mediations," tech. report RI08002, IBM, 2004.

[12] G. Lakshmanan and R. Strom, "Biologically-Inspired Distributed Middleware Management for Stream Processing Systems," in Proc. ACM Int'l Middleware Conf., Springer-Verlag, 2008.

[13] Anne Benoit, Henri Casanova, Veronika Rehn-Sonigo, and Yves Robert, "Resource Allocation Strategies for Constructive In-Network Stream Processing" Research Report 2008-20, LIP, ENS Lyon, France, June 2008.

[14] X. Gu, P. Yu, and K. Nahrstedt, "Optimal component composition for scalable stream processing," in Proceedings of 25th ICDCS, Columbus, OH, USA, June 2005.