Distributing Query Plan Operators Using Honey-Bees Algorithm

Maryam Kheirkhahzadeh
Islamic Azad university-Ahvaz branch/ Dept. of Computer Engineering, Ahvaz, Iran
Email: kheirkhah@iauahvaz.ac.ir

Mohammad G. Dezfuli
Iran University of Science and Technology, IUST / Dept. of Computer Engineering, Tehran, Iran
Email: mghalambor@iust.ac.ir

***Abstract*—Data Stream Management Systems (DSMSs) has the advantage of lightweight repositories so they can migrate in a network of hosts regarding their stream resources. We developed a framework (including some systems) based on agents to let users have their own customized query engines which can migrate in network to gain better QoS. Our Data Stream Management Agents (DSMAs) can distribute their queries and clone themselves to run different sub-queries on different hosts regarding the position of related sources. One of the main challenges toward this goal is distribution algorithm for queries. In this paper, we proposed a distributor for our agent-based data stream management framework based on honey-bees algorithm. We compare our honey-bees algorithm to other proper heuristics (genetic algorithm and DPSO) through our experimental evaluations.**

***Index Terms*— Stream, DSMS, SDMS, DSMA, Agent**

I. INTRODUCTION

Nowadays, lots of data stream generators are available and accessible through internet (e.g. sensor networks). These stream resources present promising retrievable information for world-wide users (i.e. queriers). What is missing is an infrastructure to interconnect these heterogeneous stream generation domains into a pervasive system that responds to user's queries. The transparency of this infrastructure will allow users to query it as a single unit made of millions of widely distributed ordinary or even advanced complex sensors (e.g. from thermometers to video cameras).

To have a better envision, imagine the following scenario: a biologist wishes to do research on relations between weather conditions and migrations of zebras in Kenya. Instead of travelling and living in Kenya, he can access to sensor network services deployed there and enjoy an online remote surveillance. In addition, the same resources could be queried continuously by Kenya lifeguards to detect migration of zebras toward out of the Sweetwater reserve.

We call our desired system as Pervasive Stream Information Retrieval Network (PSIRN). These five key requirements of a pervasive information retrieval network play an important role in the design of a PSIRN:

- An easy way to interconnect services and users based on available bandwidth-limited networks. First and foremost, we need an infrastructure to interconnect stream producers (i.e. services) and stream consumers (i.e. query executors). It should be robust and reliable despite unreliable low-bandwidth links.
- Finding appropriate services. PSIRN users should be capable of searching and finding services related to their queries.
- Heterogeneity of service providers and service customers. Different service providers/customers may use different customized softwares.
- Data access and privacy. PSIRN is composed of many different administration domains with different access controls and privacy policies. There should be a data access control in PSIRN that satisfies all of the domain administrators.
- Information trading and QoS. Service authors should be allowed to sell their services regarding specific QoS. In fact, an information-trading contract based on a detailed complex QoS metric [1] should be negotiated and signed by two parties.

In this paper we introduce data stream management agents which are responsible for executing a query in a data stream processing infrastructure. Each DSMA is responsible for running a subset of operators of a query plan in the network. We used JADE middleware to support relations between multiple FIPA-compliant agents. This paper is an extended version of our previous work [2] with a better focus on distribution algorithms. In this paper we used a honey-bees algorithm for operator distribution and compared it with genetic algorithm, binary PSO and centralized algorithm. We show that honey-bees algorithm performs better and is faster than other distributor algorithms.

The rest of the paper is organized as follows. First, we cover the related work in Section II. We introduce our new infrastructure in Section III. We focus on three query distribution algorithms in Section IV(honey-bee, DPSO and genetic algorithm) and demonstrate their experimental evaluation in Section V.

doi:10.4304/jsw.6.12.2464-2471

II. RELATED WORK

Aurora* [3] is a distributed version of the stream processing system Aurora [4], which assumes nodes belonging to a common administrative domain. Medusa [5] is an infrastructure supporting the federated operation of several Aurora nodes across administrative boundaries. After all, Borealis [6] is a second-generation distributed stream processing engine that inherits core stream processing functionality from Aurora* and distribution functionality from Medusa. However, none of these systems provides a flexible framework for customized query engines.

The idea of network-aware operator placement for stream processing was considered by Pietzuch et. al. in [7]. They describe a stream-based overlay network (SBON), a layer between a stream-processing system and the physical network that manages operator placement for stream-processing systems. SBON architecture uses a scalable, decentralized, and adaptive optimization technique based on a multi-dimensional metric space called a cost space. A cost space is a d-dimensional metric space where the Euclidean distance between two nodes is an estimate of the cost of routing data between those nodes. We used this distance metric and found it insufficient. They also introduced Network Usage metric and we leveraged their idea to define our Net-cost metric.

One of the areas of interest in optimization problems is swarm intelligence. It is inspired by the social behavior of some insects such as ants and bees. Honey-bees mating optimization (HBMO) is a swarm intelligence optimization algorithm that models the behaviors of bees. Honey-bees algorithms were used to model agent-base systems [8]. We used honey-bees to model our agent based query distributor algorithm. In the experimental evaluation section we show that this algorithm is better than PSO and genetic algorithm on which it is fast and tries to avoid local minima. Ref. [9,10] show independently that genetic algorithms could be elegantly useful to optimize database query plans. Ref. [11] represents a Discrete Particle Swarm Optimization (DPSO) approach for grid job scheduling. We developed a honey-bees query plan distributor and compare it to DPSO and genetic algorithm.

The efficiency and scalability of JADE and its message transportation system for large agent-based software systems has been tested independently in [12,13] and also we found it very efficient in our experiments.

III. THE INFRASTRUCTURE

To overcome low bandwidth and unreliable links, we used mobile query processors. It means we can get a part of query plan to a mobile query processor and then this processing unit can go to an optimal position that is the nearest position to its needed stream sources. On the other hand, mobility is not possible for heavyweight DBMSs because of their high volume data but it is good for lightweight DSMSs (that contain only queries and some data sketches). Thus we need a multi-agent system because each service provider or service customer works on behalf of its owner. We also need a layer to simplify wide access to data stream sources that supports service oriented architecture (SOA), compatible with internet, supports different kinds of DSMSs and supports mobility of DSMAs.

Our PSIRN is based on agent-oriented java-based JADE middleware [14,15]. JADE is a framework to develop multi-agent systems in compliance with FIPA specifications [16]. JADE could be run on heterogeneous platforms from powerful servers to cell phones. It also supports mobile agents and yellow pages (to find service providers) and facilitate message passing between agents.

It has an acceptable performance and scalability even for heavy loads [12,13]. We have made the simplifying assumption that there is only one administration domain and we leave the security, QoS and information trading for future work.

A. The Architecture

Each site runs a middleware whose major components are shown in Fig. 1, 2. The architecture is based on four types of agents:

1) Data Stream Management Agents (DSMAs)
2) Global Service Directory (GSD)
3) Stream Proxies (SPs)
4) Topology Explorers (TEs)

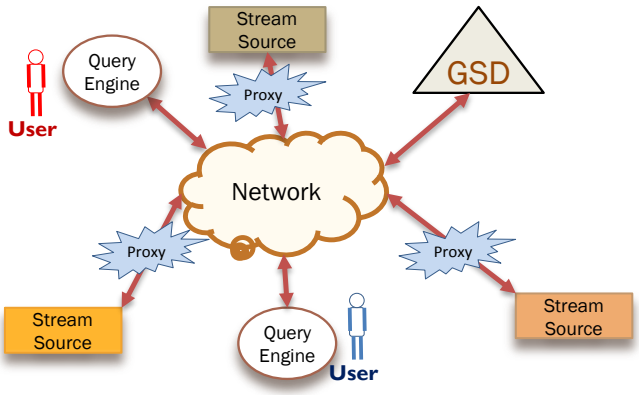

Figure 1. The Topology

DSMAs are agent-based DSMSs. The DSMA is a mobile agent containing a pre-designed and tested DSMS engine.

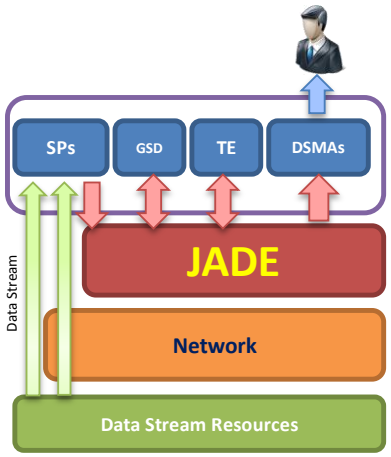

Figure 2. The Architecture

The GSD is an immobile agent that stores information about services. Users can find proper services using GSD. One GSD is sufficient for an administration domain. The SPs are customized interfaces between stream sources and service customers. Each SP is well adapted to a stream source and should register its services in the GSD and provide service for DSMAs in a standard pre-defined manner. In fact, SPs are mobile agents and they travel to the best host regarding their stream sources. There is one immobile TE for each host. TEs compute the network delays between their host and other hosts. TEs help each other to make a Virtual Network Model (VNM) based on peer to peer network delays. TEs update This VNM without any interaction with network layer and it helps other components to look at network layer as a black box. VNM is essentially needed for optimal query distribution in DSMAs.

B. The DSMA Architecture

The DSMA architecture is illustrated in Fig. 3. Some modules like Query Plan Manager, Memory Manager, Scheduler, QoS Manager and AAA (Authorization, Authentication and Admission Control) are similar to their counterparts in DSMSs. The GUI helps user to search and find proper data sources and enter queries. It also displays the results of the query. The JADE Interface Module is an interface between DSMAs, GSD, SPs and TEs. It handles the message passing between mentioned agents. Input ports of DSMA should be connected to JADE interface and output ports could be connected to a file, GUI or JADE interface. Query Decomposer is the most complex and important module that we added to DSMA. Query Decomposer is responsible for decomposing queries using some QoS metrics and network model. It requests Migrator to send different cloned DSMAs to selected hosts. Migrator is responsible for migration of these new generated DSMAs.

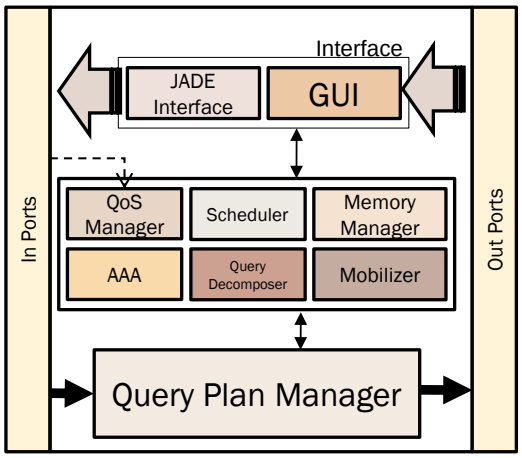

Figure 3. The DSMA Architecture

C. The Connections between Agents

We used Internal Message Transport Protocol (IMTP) of JADE to interconnect agents. We also used XML instead of the JADE's ontological approach for our messages. Agents use yellow pages service of the JADE to find basic services like the GSD.

JADE names each agent with a globally unique identifier. It also can automatically handle the mapping between name of the agent and its physical address. JADE does this mapping using a mapping table in the main container. Other containers use cached mapping table to speed up the mapping process. It helps to prevent any bottleneck in the main container. Agents can communicate directly through their containers and without interference of the main container. Network layer is responsible for low-level message routing.

IV. QUERY DISTRIBUTION

The most important and complex part is how to decompose queries and distribute sub-queries on the network. DSMAs distribute query plans regarding VNM and the locations of producers and customers. Similar cloned DSMAs migrate to selected hosts carrying different parts of the decomposed query. There are two important objective functions to achieve an optimal query distribution: 1) minimizing network usage and 2) minimizing response time of queries.

There are three challenges when facing the query distribution problem: 1) modeling topology of the network (i.e. VNM), 2) decomposing queries into some sub-queries and 3) sub-query placement. The second and third challenges are toward an objective function and are tightly related so they could be covered by a single method (as we did).

A. Virtual Network Model

Our infrastructure lies upon the network layer and treat it as a black box. This helps having an infrastructure for many different interconnected networks and it is necessary for internet infrastructures. Because of the hidden aspects of the underlying network, we need a virtual network model to achieve a near optimal query distribution. The VNM should be creatable without any help from the network layer.

We model network as a complete weighted graph. In this graph, the communication delays between hosts (i.e. vertices) are modeled as weights of the edges (i.e. virtual connection links). This model is proper for different mentioned cost functions (i.e. two objective functions for optimization). In addition, middleware can generate and update this model independent of underlying network layer. The only drawback of VNM is high cost of updating weights of the edges (it has a complexity of $O(n^2)$ where n is the number of hosts). We can improve the cost using minimum spanning tree to bring the complexity down to $O(n)$. Inter-domain distribution is not allowed (because of the policies of administration domains) and sub-queries should be sent to administration domains to be distributed by them locally. Thus, the current updating cost is acceptable for us.

TEs are responsible for generating and updating VNM. They do it using permanent peer-to-peer messaging. DSMAs can get the model from main TE of the administration domain and use it to get a near-optimal decomposition and distribution.

B. Modeling Query Plans

Query plans could be derived from registered continuous queries. They are composed of operators, which perform the actual data processing; queues, which buffer data as it moves between operators; and synopses, to hold state of operators.

Operators are the basic data processing units in a query plan. An operator takes one or more streams as input and produces a stream as output. As in a traditional DBMS, a plan for a query connects a set of operators in a tree: The output of a child operator forms an input of its parent operator, the input streams and relations of the query form the input of the leaf operators, and the output of the root operator forms the output of the entire query. Because of the variety of operator types, we model operators as black boxes with multiple inputs and outputs. For each operator O, D(O) is the processing delay for each input tuple and R(O) is the ratio of output to input (rate of output generation).

C. Query Decomposition and Distribution

Distributing *m* operators among *n* hosts is a NP-hard problem. We developed our distributor based on honey-bees, PSO and genetic algorithm and compared them to each other. We show that honey-bees is the best one. In this section we explain in brief each algorithm and its encoding schemes that we used. Encoding is a mapping from knowledge domain to the solution space where our algorithms can process. The selection of encoding scheme is varied with the design decision and also depends on the problem to be solved. In all algorithms, we model each solution by a $1 \times n$ array. *n* shows the number of operators in the query plan. Each cell *i* of the array contains a host number that we assigned *i*th operator to it.

C.1 discrete PSO

Particle swarm optimization (PSO) is a population based stochastic optimization technique introduced by Kennedy and Eberhart [17]. PSO simulates social behavior of bird flocking. The algorithm is initialized with a population of random solutions. Each solution represents a particle. All particles move based on pbest (personal best) and gbest (global best) in the search space to find the best location. $pbest_i$ is the best location that *i*th particle has experienced so far. gbest stands for global best and is the best location that all particles have experienced so far. Pbest and gbest should be updated after each iteration of the algorithm. The algorithm repeats until a threshold is reached or it finds the optimal solution. In fact, after each iteration, the position of each particle updates with the velocity vector. Velocity vector is calculated based on pbest and gbest.

PSO is applicable in many fields such as function optimization, artificial neural network training, fuzzy system control. There are two versions of PSO algorithms. The basic PSO is suitable to solving continuous problems. The second version named binary PSO is capable of solving discrete problems and introduced by Kennedy and Eberhart [18]. Similar to the work in [18], we used binary PSO to solve our discrete optimization problem because our goal is to assign proper host numbers to the operator set and the search space of our algorithm is discrete instead of continuous (domain of host numbers is a discrete set). We also used DPSO [11] algorithm that is a discrete version of PSO and it has a high performance.

C.1.1 Encoding Scheme

Izakian et. al. proposed an efficient discrete PSO algorithm with a direct representation named DPSO [11]. We can use two representations for a discrete PSO problem to encode each solution: 1) indirect and 2) direct. In indirect representation, each particle must be modeled by a two dimensional vector $h \times n$. *h* shows the number of hosts and *n* is the number of operators. If the operator *i* is assigned to the host number *j*, the proper cell should be one, otherwise zero. That means cells show what operator has assigned to which host by the one or zero value. However, direct representation uses a $1 \times n$ vector for each particle which *n* is the number of operators and each cell has an integer value shows the number of host that assigned to that operator. We used efficient direct representation DPSO algorithm. We chose global best instead of local best to speed up the algorithm. Each particle modeled as a $1 \times n$ array. Gbest and pbest are also $1 \times n$ arrays. We modeled velocity vector, with a $p \times h \times n$ array. The *p* parameter is the number of particles, *h* is the number of hosts and *n* is the number of operators in the query plan.

C.1.2 distribution algorithm

Fig. 4 shows the discrete PSO distribution algorithm that we used. PSO algorithm first generates initial random particles and then assigns each particle to its pbest. After that, it assigns the best pbest to the gbest. In the *while* loop in step 2, PSO calculates the fitness value of each particle and then updates pbest and gbest. Thus, PSO can calculate velocity vector of each particle by updated gbest and pbest. Note that our velocity updating function is like DPSO[11]. Finally, new particles can be generated using new velocity vectors.

```
1.      Initiate random particles and assign
        each particle to its pbest.
        Find initial global best.
2.      While have enough time
2.1.      Calculate fitness of particles
2.2.      For each particle Update pbest
2.3.      Update gbest
2.4.      For each particle do these steps
          with DPSO algorithm
2.4.1        Update its velocity vector
2.4.2        Update its position
3.      Return gbest particle as solution
```

Figure 4. Discrete PSO based distribution algorithm

C.2 Honey- bees algorithm

The idea of honey-bees algorithm is to simulate the behavior of honey bees. There is a queen in the honey bees' colony. Queen starts a mating flight with an initial speed and energy and mates with drones. After each

mating flight, the speed and energy of queen will be decreased. Queen returns home when its speed or energy is near zero or queen's sperm repository is full.

Honey-bees algorithm is composed of genetic, simulated annealing and local search algorithms. It uses simulated annealing for selecting best chromosomes in the selection phase of genetic algorithm. Using simulated annealing in the honey-bees algorithm reinforces it to escape from local minima. It is an important characteristic of simulated annealing algorithm because it also accepts worse solutions with a probability. We show in our experimental evaluation that honey-bees does not get trapped in local optima. Other benefit of using simulated annealing in the selection phase is that only the strongest drones will be selected so the number of times the fitness function invoked will be decreased. The probability of selecting each drone in honey-bees algorithm determined by (1).

$$P(Q,D) = e^{-ABS(\frac{f(Q)-f(D)}{speed})} \quad (1)$$

In the above formula, *f* is a fitness function. *f*(Q) is fitness value of queen and *f*(D) is fitness value of drone. The speed of queen decreases after each iteration of the algorithm so the selection probability for a drone in the initial steps is more than next steps. The difference between *f* (Q), and *f*(D) is another effective factor. In fact, the smaller the difference, the more the selection probability.

C.2.1 Encoding scheme

We modeled each drone and each brood by a $1 \times n$ array. We also defined queen as a $1 \times n$ array.

C.2.2 Distribution algorithm

Fig. 5 shows our honey-bees distribution algorithm. At first, initial population generator function gets the hive size (the number of initial drones) as the input parameter. and generates random initial drones. It also assigns the best one to the queen. The number of queens may be more than one. However, we decided to have one queen in our algorithm.

Ref. [19,20], used random initial values for the initial speed and initial energy of the queen, in step 2. We set the initial speed value to $5e^{10}$ like [21], because it leads to generate better results in our experiments. We used random initial value for energy. While energy is more than zero, we select a drone and then based on its selection probability, which is computable using (1), it may be selected and added to the queen's repository for making a brood. After selection step, we should crossover each selected drone with the queen to make a new brood. Note that queen does not change. The result of our crossover algorithm is an $1 \times n$ array as a new brood. Our crossover algorithm, selects randomly p_1 percent of operators from selected drone and replaces their host numbers with the proper host numbers of the queen. At the next step, mutation function works on the new brood. The role of mutation is to keep the diversity of population. We define two kinds of mutations. First mutation algorithm chooses two random operators from the new brood and swaps their host numbers with each other. Our second mutation algorithm chooses a random operator from the new brood and simply assigns another random host number to it. We named our first mutation function *Mutation1* and the second one, *Mutation2*. We tested two types of our mutation functions and found out that *Mutation2* is more efficient for our honey-bees algorithm and we used it. However you can see in the next section that *Mutation1* is more suitable for our genetic based distribution algorithm.

In step 2.5, if a new generated brood exists that has a better fitness than queen, queen must be replaced with it. Other broods should be saved for the next generation.

In the final step, We kill and remove old drones and make new ones for next generation. Our new generation contains the new broods generated in current iteration plus new randomly generated drones like [19].

```
1.      Generate initial random drones(Hivesize)
        Assign the best drone to the Queen
2.      While have enough time
2.1.      Speed=5e^10
2.2.      Energy=rand[0.5,1];
2.3.      While energy>0&& Queen's Repository
          isn't full
2.3.1.      Select next drone and calculate Δ
            (Δ(f)=fitness(Queen)-fitness(drone))
2.3.2.       Generate r=rand(0,1);
2.3.3.      If(exp(-|Δ(f)/speed|)>r)
              Add current drone to Queen's
              Repository
              Update speed and energy
2.4.        For each drone in Queen's Repository
2.4.1.        crossover drone with the Queen
              with p1%relocation to make a brood
2.4.2.        use Mutation1 for new brood.
2.5.        For each brood do
              If fitness(brood)>fitness(Queen)
                        Queen=new brood
              else
                        Save the new brood
2.6.        Generate new drones by new broods
            plus randomly generated drones
3.      Return queen as solution
```

Figure 5. Honey-bees distribution algorithm

C.3 Genetic algorithm

Genetic algorithms are stochastic search methods based on natural biological evolution and also they are in class of global search methods. J. H. Holland in [22] did much work to develop genetic algorithms. They have been applied to a wide range of optimization problems, including well-known NP-complete and NP-hard problems, scheduling and routing, configuration, and query optimization [9,10].

In genetic algorithms, each solution is modeled as a chromosome and a collection of these chromosomes is called a population. They start with a population that usually generated randomly or may be heuristically. In each iteration, some of the best chromosomes would be selected according to their fitness values. Genetic algorithms have two operators: crossover and mutation.

After the selection phase, the crossover operator tries to make better chromosomes out of selected ones. The goal of mutation operator is to increasing the diversity by applying random changes into the chromosomes.

C.3.1 Encoding Scheme

Encoding scheme will affect the selection of genetic operators. Improper encoding scheme will produce infeasible chromosomes generated by genetic operators. Usually, chromosomes are represented as fixed length strings coded with a binary character set. Other types of encoding schemes include real number representations and permutation representations. We model each chromosome by a $1 \times n$ array. n is the number of operators in query plan and each cell shows the related host that operator assigned to it.

C.3.2 distribution algorithm

Crossover is the most important operator in genetic algorithms. It takes valuable information from both parent chromosomes and then combines them to find a highly fit chromosome. Our crossover algorithm simply selects p_3 percent of operators randomly from a chromosome and then changes their hosts with the hosts in another randomly selected chromosome.

We used *Mutation1* for our mutation function in genetic algorithm. As we explained in section C.2.2, it chooses two random operators from current chromosome and simply swap their host numbers to each other. We tested two types of mutation and found out that this type of mutation is more efficient for our genetic based distribution algorithm. Fig. 6, shows our genetic based distribution algorithm.

```
1.    InitRandomChromosomes (p1 samples)
2.    While have enough time
2.1.    Calculate fitness of chromosomes
2.2.    Sort chromosomes regarding their
        fitness
2.3.    Select p2 number of the best
        chromosomes
2.4.    CrossOver each chromosome with a
        random chromoshome with
        p3% relocations
2.5.    Mutate each chromosome with
        probability of p4 (Mutation1)
3.    Return chromosome with best fitness as
      solution
```

Figure 6. Genetic-based distribution algorithm

C.4 fitness Function

To evaluate our results we need to determine one or more fitness functions. We used two fitness functions: ART-Cost and Net_Cost. Fitness function gives each solution a fitness value which is a judgment of its surviving capability. Choosing and formulating an appropriate fitness function is crucial in obtaining efficient solution for those problems solved by the algorithms.

ART-Cost is the ratio of the average response time to the ideal response time. We compute the ideal response time regarding two conditions: 1) operators work concurrently with maximum delay, and 2) no delay for interconnecting links. For example, ideal response time for query plan in Fig. 7, is equal to 22.

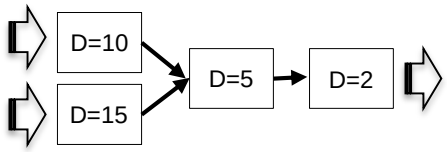

Figure 7. An example for query plan

Net-Cost is the bandwidth-delay product of the query. Net-Cost captures the idea that the longer the data stays in the network, the more likely it is to traverse nodes and links that could be used for other queries [7]. The Net-Cost for a query q, is the amount of data that is in-transit for q at a given instant and calculates by (2).

$$u(q) = \sum_{l \in L} B(l)L(l) \qquad (2)$$

Where L is the set of links used by q, B(l) is the bandwidth of link l, and L(l) is the latency. We can compute fitness function based on these two cost models using approximating (determined statistically) network and query parameters.

V. EXPERIMENTAL EVALUATION

In this section, experiments are conducted to testify the advantages of our proposed approach in terms of Net-cost and ART-cost metrics. All algorithms in the experiments are implemented with java language. We used two simulation-based scenarios to evaluate our distribution algorithm. We compared honey-bees, DPSO and genetic algorithm with each other and with centralized algorithm for each scenario. The centralized method reveals the effect of distribution.

The first scenario is about using high fan-in query plans [23]. We used low fan-in query plans (just like binary trees) for second scenario.

First we show how the fitness value of each distribution algorithm changes over time and also you can see in Fig. 8, that honey-bees algorithm does not get trapped in local minima.

For each operator O, D(O) and R(O) are both equal to 0.5. For honey-bees algorithm, we defined one queen. We selected the initial value of $5e^{10}$ for speed and a random value in range (0.5,1) for energy. Decreasing rate for energy and speed is 0.9. To select the size of queen's repository, we tested lower values than 4 and greater values than 4 and we found that 4 is the best one. Also hive size is set to 20. We set the crossover probability p_1 to 60. In the PSO algorithm, we set $c_1=c_2=2$, $r_1=r_2=1$, $v_{max} = 4$. In addition, in the genetic algorithm $p_1=100$, $p_2= 50$, $p_3=0.6$ and p_4 is equal to 0.3. The data sources and users are hosted by random hosts.

A. Relative changes in values of fitness over time

We executed each algorithm 100 times. Fig. 8, represents changes in average fitness value over time interval (0,40). The Fitness function is Net_cost in high fan-in scenario. You can see that PSO and genetic algorithm get trapped in local minima and fail to reach the goal but honey-bees algorithm performs well because of the benefit of using simulated annealing in its selection phase. In fact, simulated annealing tries to avoid local minima. In addition, Fig. 8, represents that honey-bees algorithm is considerably faster than genetic and PSO algorithms. The selection phase of honey-bees algorithm speeds up it because of selecting strongest drones to mate with queen. Therefore, the number of fitness function, crossover and mutation invocation will be limited to the size of queen's repository.

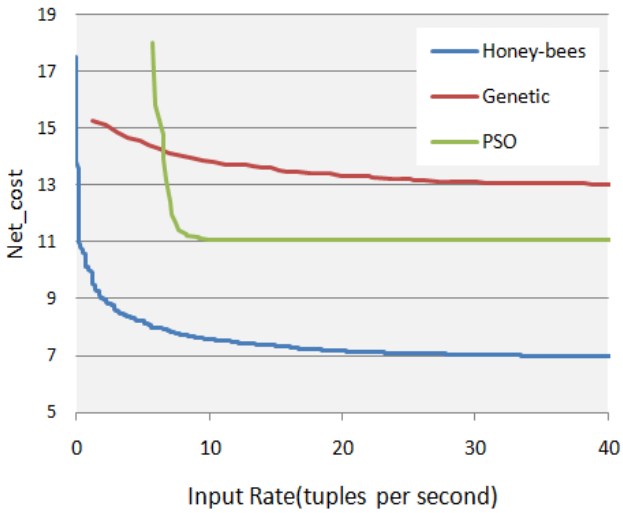

Figure 8. The values of fitness at the first 40milisecond

B. First Scenario: High Fan-in Query Plans

In this scenario, the network model composed of 400 randomly placed hosts. The query plan is a tree with depth of 3 and fan-in of 7 (i.e. each operator has 7 inputs and 1 output). So each query plan has 393 connected operators.

The chart in Fig. 9, shows how Net-cost value changes almost linearly as the input stream rate in the system is increased. In this experiment, we limit the execution time of distributors to 0.8 minute. The effect of changing R and D parameters is just like input rate so we ignore it. Our query distributors are better than centralized algorithm. Honey-bees algorithm is the best distributor and it has a lower fitness value in each step than other algorithms. The results for PSO and Genetic algorithm are almost like each other.

The obtained results in Fig. 11, show how better our honey-bees distributor is in ART-cost metric in respect to the other distributors. Centralized algorithm has a bit lower fitness than honey-bees algorithm. It is trivial because in the centralized algorithm all operators are assigned to one host so the cost of network delay and bandwidth will be decreased. Note that the values of R and input rates are not effective in ART-Cost metric.

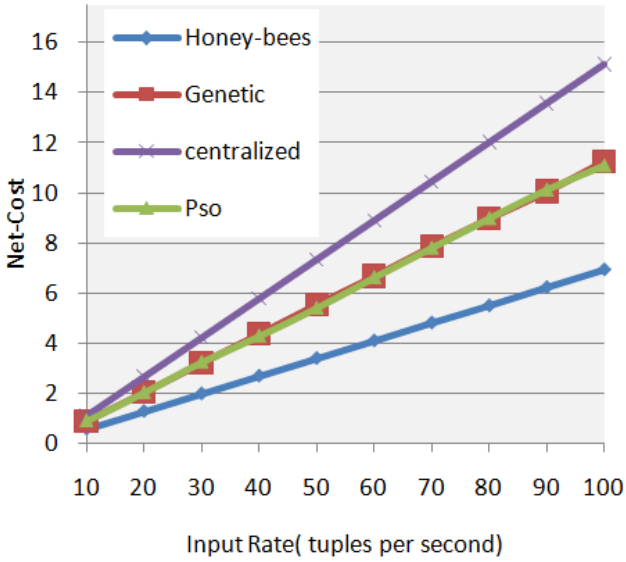

Figure 9. Net-cost for high fan-in queries

C. Second Scenario: Low Fan-in Query Plans

In this scenario, we used usual low fan-in query plans with depth of 2 and fan-in equals to 2 (i.e. each operator has 2 inputs and 1 output). So each query plan has 2 connected operators. Other parameters are just like the previous scenario.

The charts in Fig. 10, 12, show that all algorithms almost operate like each other. Using query plan distributors in low fan-in scenario does not lead to better Net_cost. Hence, distribution does not play an important role in low fan-in scenario. In high fan-in scenario, the centralized algorithm no longer performs well.

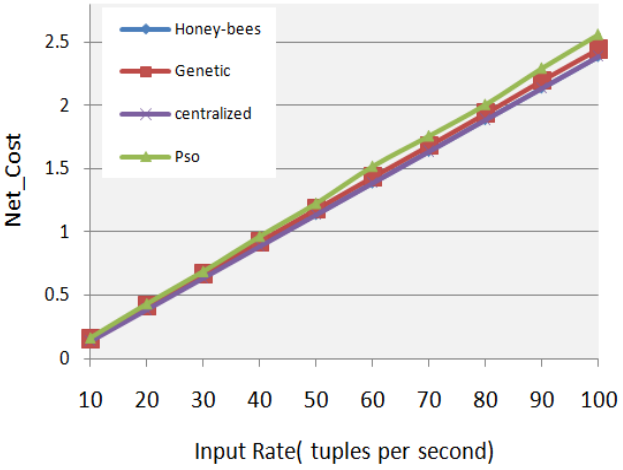

Figure 10. Net-cost for low fan-in queries

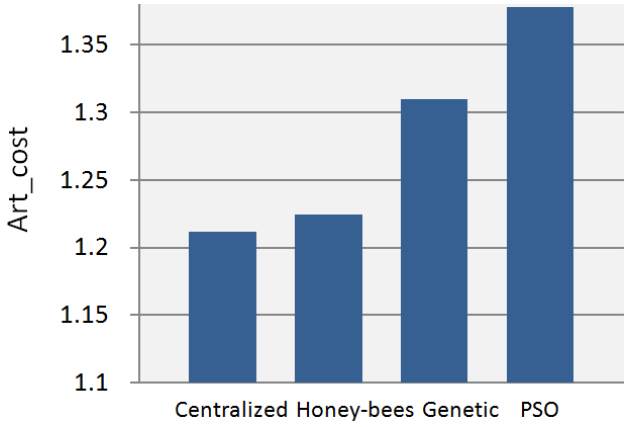

Figure 11. ART-cost for high fan-in queries

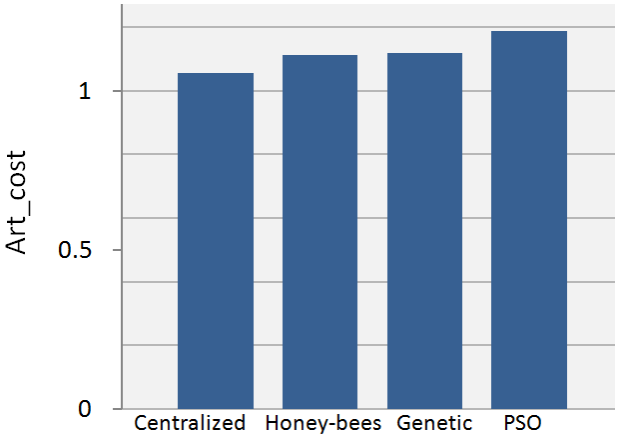

Figure 12. ART-cost for low fan-in queries

VI. CONCLUSION

We introduced a new framework for data stream query processing which is based on an Agent-based middleware, JADE, and allow users to have agent-based query engines which are capable of decomposing their queries to sub-queries and clone themselves to distribute query processing. This idea helps us to process each sub-query near its stream sources and leads to less net costs.

The most challenging and complex module in our systems was query distributor. We implemented our query distributor system with honey-bees algorithm and compared it to DPSO and genetic algorithm. DPSO is the binary version of PSO and suitable for discrete domains. In the experimental evaluation section we showed that our honey-bees based query distributor algorithm is more efficient than two other distributors (esp. for high fan-in queries) and it increases the network utilization.

REFERENCES

[1] M. Ghalambor, A.A. Safaeei, and M.A. Azgomi, "DSMS scheduling regarding complex QoS metrics", IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), 10-13 May 2009.

[2] M. Kheirkhahzadeh and M. G. Dezfuli, "Agent-based pervasive stream information retrieval", In proc. Of the IASTED Int. Conf. on Artificial Applications, AIA 2011, pp.171-176, Innsbruck, Austria, Feb 14-16, 2011.

[3] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, et al., "Scalable distributed stream processing", In Proc. of the 1st Biennial Conference on Innovative Data Systems Research (CIDR), CA, January, 2003.

[4] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, et al., "Aurora: a new model and architecture for data stream management", The VLDB Journal The International Journal on Very Large Data Bases, Vol. 12, No. 2, pp. 120-139, 2003.

[5] M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Load management and high availability in the Medusa distributed stream processing system", In Proceedings of the ACM SIGMOD international conference on Management of data, NY, USA, 2004.

[6] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, et al., "The design of the Borealis stream processing engine", In Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, January, 2005.

[7] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, et al., "Network-aware operator placement for stream-processing systems", In Proc. of ICDE, April, 2006.

[8] A. Pérez-Uribe and B. Hirsbrunner,"Learning and foraging in robot-bees", in Meyer, Berthoz, Floreano, Roitblat andWilson (eds.)', SAB2000 Proceedings Supplement Book, Intermit. Soc. For Adaptive Behavior, Honolulu, Hawaii, pp. 185–194.

[9] M. Stillger, and M. Spiliopoulou, "Genetic programming in database query optimization", In Proc. of the Genetic Programming Conference, pp. 388-393, July 1996.

[10] K. Bennett, M.C. Ferris, and Y.E. Ioannidis, "A genetic algorithm for database query optimization", In Proc. of the 4th International Conference on Genetic Algorithms, 400-407, 1991.

[11] H. Izakian , B. Tork Ladani, A. Abraham, V. Snasel, "A discrete particle swarm optimization approach for grid job scheduling". International Journal of Innovative Computing, Information and Control vol .6, No. 9, September 2010.

[12] E. Cortese, F. Quarta, G. Vitaglione, and P. Vrba, "Scalability and performance of JADE message transport system", Journal of Analysis of Suitability for Holonic Manufacturing Systems, Vol. 3, No. 3, pp. 52-65, 2002.

[13] K. Chmiel, "Efficiency of JADE agent platform", Scientific Programming Journal, Vol. 13, No. 2, pp. 159-172, 2005.

[14] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE--a FIPA-compliant agent framework", Proceedings of PAAM, Vol. 99, pp. 97-108, 1999.

[15] JADE Official Website, http://jade.tilab.com.

[16] P.D. O'Brien, and R.C. Nicol, "FIPA — towards a standard for software agents", BT Technology Journal, Vol. 16, No. 3, pp. 51-59, Jul 1998.

[17] J. Kennedy and R. C. Eberhart, "Particle swarm optimization", Proc. of the IEEE International Conference on Neural Networks, pp.1942-1948, 1995.

[18] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm", IEEE International Conference on Systems, Man, and Cybernetics, Orlando, FL, vol.5, pp.4104-4108, 1997.

[19] N. R. Sabar, M. Ayob, G .Kendall,"Solving Examination Timetabling Problems using Honey-bee Mating Optimization (ETP-HBMO)", (MISTA 2009), August2009.

[20] O. B. Haddad, A. Afshar and M. A. Marino, "Honey-Bees Mating Optimization (HBMO) Algorithm:A New Heuristic Approach for Water Resources Optimization", Water Resources Management (2006) 20: 661–680,DOI: 10.1007/s11269-005-9001-3.

[21] O. B.Haddad, M. Mirmomeni,M. Z. Mehrizi,M.A.Mariño ,"Finding the shortest path with honey-bee mating optimization algorithm in project management problems with constrained/unconstrained resources", Published in Journal of Computational Optimization and Applications, Volume 47, Issue 1, September 2010.

[22] J. H. Holland. "Adaptation in natural and artificial systems", Ann Arbor:University of Michigan Press, 1975.

[23] M.J. Franklin, S.R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, et al., "Design considerations for high fan-in systems: the HiFi approach", In Proc. Of the CIDR Conf., Jan. 2005.