Scavenging Complex Genomic Information Using Mobile Code: An Evaluation

Mauro Pezzè, Davide Tosi
University of Milano-Bicocca
Via Bicocca degli Arcimboldi, 8
I-20126 - Milano, Italy
{pezze|tosi}@disco.unimib.it

Gian Pietro Picco
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 - Milano, Italy
picco@elet.polimi.it

Abstract

Mobile code solutions can improve the performances of applications over the Internet by reducing the amount of data that must be transferred across the network. The research on genome examines enormous amount of data stored on many distributed databases to retrieve the information relevant for the specific investigation, and thus is likely to greatly benefit for mobile code solutions.

This paper studies the benefits of mobile code solutions for research on genome, by comparing different solutions with analytical models, and presenting the results of case studies that allow to identify when mobile code overwhelm client/server solutions. The measured improvements of performance indicate that the retrieval of information from genome databases is a promising domain for mobile code applications.

1 Introduction

The World Wide Web enables access to enormous amounts of heterogeneous information dynamically distributed over the Internet. When the information of interest is not immediately identifiable, but requires non-trivial elaboration, traditional client/server applications can generate an enormous amounts of information flow that can imply a significant transmission time.

Retrieval systems based on the mobile code paradigm can largely reduce transmission time by elaborating the data remotely, and thus reducing the amount of information that needs to be transferred [12, 7]. Mobile code solutions have been evaluated in many domains, e.g., e-commerce [9], document retrieval [8], wireless networks [10], internetwork management [3]. The speed of the network often mitigates the end user perception of the performance differences between client/server and mobile code solutions that are perceivable only when the amount of data remotely filtered is extremely large. In most cases, we can experience a reduction of the traffic on the network, but not a perceivable reduction of the end-to-end delay.

The research on genome is producing enormous amounts of heterogeneous data dynamically distributed over the Internet that grows at an amazing speed and requires complex elaboration to be turned into useful information for genomic research. Today biologists rely mostly on classic client/server systems with dreadful performances [5]. The introduction of federated databases [1] and datawarehousing [16] alleviates but does not completely solve the problem. Statistical predictors [15] and text mining mechanisms [6] improve access to the scientific literature, but do not help accessing other essential information in the field, e.g., genomic sequences.

The amount of time that biologists waste in transferring data for local elaboration suggests that the search for genomic information may greatly benefit from mobile code solutions. In this paper, we discuss potential benefits of mobile over classic solutions through simple analytical models, and we assess the predicted benefits though a set of cases studied with prototypes that allows us to experiment with client/server, remote evaluation and mobile agent solutions on WAN, LAN and ADSL networks interfacing popular heterogeneous databases. The paper witnesses experiences in performance reductions up to a factor of 17 for ADSL networks that can reduce search time from hours to minutes.

The main contributions of this paper are: the identification of the research on genome as an important application domain for mobile code solutions, the analytical predictions of the benefits of mobile code solutions for different classes of queries typical of the considered application domain, the outline of prototype tools for experimenting with different paradigms in different situations and the discussion of case studies that show the benefits of remote evaluation and mo-

bile code solutions over client/server solutions for different classes of queries, thus identifying the areas where advantages are particularly relevant.

The paper is organized as follows. Section 2 summarizes the genomic information retrieval problem and surveys the state of the art in the area. Section 3 identifies remote code evaluation and mobile agents as the most promising mobile approaches to this problem, and analytically compares the performances of these approaches with classic client/server solutions for different kinds of queries. Section 4 briefly describes the prototypes used for the experiments and motivates the plans of the experiments. Section 5 discusses the obtained results, and provides early quantitative evidence of the potential advantages of mobile over client/server solutions for different classes of queries that characterize the applicative domain. Section 6 concludes summarizing the main results presented in the paper and indicating future research directions.

2 The Genomic Information Retrieval Problem

The research on genome is producing an enormous amount of information that is scattered in many databases and is growing with exponential rate. For example, in July 2004 the popular *PubMed* [`http://ncbi.nlm.nih.gov/entrez/`] database contained more than 14.000.000 entries, and a simple query with the term 'HIV' returned approximatively 135.000 entries. Today PubMed contains many more entities and the same query would return many more results. Size and growth of the information available over the Internet is only one of the problems that biologists must face. A lot of genomic information is non-homogeneous, redundant and badly structured, since it often derives from experiments that provide only partial data and is stored in proprietary formats. Thus, biologists must query many databases and compare the partial data extracted from different sources to improve the quality of the needed data [6]. The retrieved information must be filtered to eliminate redundancy and integrate incomplete data. For example, a search for the genome sequences related to a form of life, e.g., a virus, on many databases, can return many sequences partially overlapping and partially complementing each other. Scientists may be interested only in the most complete of the currently available sequences, which can be obtained by discarding many redundant data, and keeping only the most complete complementary sequences that must be suitably combined. Filtering often removes an enormous amount of information that is responsible for long search time.

The poor performances of traditional client-server tools [5] have been improved by introducing federations of databases built with hyperlinks that allow clients to search though several databases with a single query. Federative solutions are provided for example in the SRS Sequence Retrieval System [1] and in the Entrez System [21]. The advantages of these approaches are limited by the unidirectionality of the hyperlinks and by the lack of incremental filters. The unidirectionality of the hyperlinks limits browsing capabilities, while lack of filters does not reduce the enormous amount of information that grows when querying many databases.

Better solutions are based on data-warehousing, as for example in the PROSITE system [16]. These solutions improve the access to multiple databases and reduce the problems that derive from language heterogeneity, but do not address the problem of information redundancy.

The information redundancy problem is partially addressed by Ponte and Croft [15], and by Feldman et al. [6] who focus on retrieving information in the form of reports or papers but do not adapt to other kinds of data. Ponte and Croft propose a statistical model to estimate the probability that a document could be relevant to the query; while Feldman et al. use data and text mining mechanisms to automatically select relevant literature [6].

All the solutions proposed so far are far from being optimal and biologists waste a lot of time in retrieving and filtering a huge amount of redundant information.

3 Mobile Code Solutions for the Genome Retrieval Problem

Code mobility, that is the ability to relocate dynamically the components of a distributed application [14, 7, 4], seems to well address the different aspects of the genomic information retrieval problem. The possibility of migrating code across locations allows both the search in many databases for answering a single query, and the reduction of data transfer by locally filtering the retrieved information. Moving code increases the possibility of querying databases in parallel, since different agents that serve a single query can be executed concurrently on different databases when the results of the searches are not mutually dependent.

3.1 Architecture

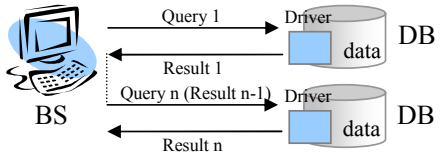
Code mobility indicates many approaches that can be classified as code on demand, remote evaluation and mobile agents. All paradigms operate in a distributed environment where an application running on a node (referred to as base-station), queries several databases for retrieving the information of interest. In this paper we evaluated the remote evaluation and mobile agent solutions.

In the remote evaluation paradigm (Figure 1a), the client machine owns the code for elaborating the results, while the

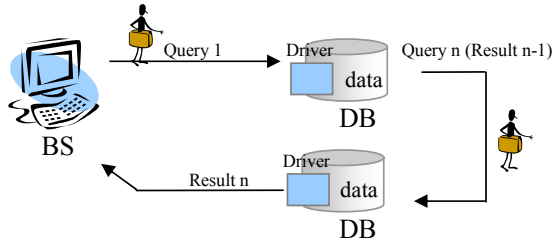
database machines own the data. The client sends the code to the database machines, which execute it on the local data, and send the results back to the client machine. The code for elaborating the results is composed of a generic driver, that is a program to extract and filter data from a database, and a computational logic, that is the specific computation required by the clients to complete the required searches. Since databases are involved in many different computations that rely on the same generic drivers, the generic drivers can be cached on the databases, and the clients can send only the computational logic.

In the mobile agent paradigm (Figure 1b), the agents migrate to the databases carrying the programs for remote filtering and some intermediate results and states. As in the case of remote evaluation, the programs include generic drivers that can be cached on database machines to reduce the size of the agents that migrate across the network.

Both remote evaluation and mobile agents seem appealing solutions since they both filter data on the remote databases, thus substantially reducing the amount of data exchanged over the network. Moreover, they both allow for parallel execution of independent code on different databases. In our study, we assume that generic drivers are cached on the database machines, and thus we send only computational logic and data across the network.



(a) Remote Evaluation: The code migrates without state and is executed on the remote databases (DB).



(b) Mobile Agent: The code migrates with the associated state and is executed on the remote databases (DB).

Figure 1. Code Mobility Paradigms

3.2 Analytical Model

The benefits of mobile code solutions can be studied analytically. Here we examine the mobile agent and the remote evaluation solutions, and compare them with the classic client/server solution.

The analytical models depend on the kind of query that can be single, incremental on a single database, incremental on several databases and independent on several databases. A single query involves a single access to a single DB. For example searching proteins related to HIV on a specific DB. Incremental queries iteratively access a single DB to incrementally refine or enrich the results. For example, we can first search papers about HIV, then restrict the search to papers about a specific set of proteins among the ones retrieved in the first step, and then further restrict the search looking for papers that contain genomic sequences among the ones formerly retrieved, and so on. In incremental queries, the keys of the successive steps are computed from the results of the former steps and thus cannot be included in a single query.

All models estimate the total response time T_{tot}, i.e., the time between the formulation of the query and the availability of the results in the base-station. T_{tot} is computed as a function of several parameters that express the dimension of the data exchanged over the network, the execution time of the programs involved in the computation, the speed of the network and of the involved nodes, the network latency τ defined as the time it takes for a packet to cross a network connection, from sender to receiver (measured in ms).

Data exchanged over the network (measured in bits)

D_{query} indicates the size of the query. It influences the transmission time of the query from the base-station to the target databases in the case of client/server solutions.

D_{rev} indicates the cumulative size of the query and of the associated code in the remote evaluation solution. It influences the transmission time of the query and the program from the base-station to the target databases in the case of remote evaluation solutions.

D_{agent} indicates the size of the serialized mobile agent. It includes the size of the agent, its status and the plan of the query, and influences the transmission of the agent between the base-station and the databases in the case of mobile agent solutions.

D_{res} indicates the size of the results and determines the transmission time to the base-station in all cases.

Execution time of the involved programs (measured in ms)

T_{db} indicates the response time of the database, computed as the time between the start of the elaboration of the query by the database and the production of the results.

T_{str} indicates the time for converting the stream of results as produced by the database into a string suited for transmission over the network.

T_{ser} represents the agent serialization time, i.e., the time needed to serialize the agent for moving it to a different node.

Execution time of the involved programs

DB indicates the speed of the machine hosting a database, and determines the time for filtering the results in the case of mobile solutions.

BS indicates the speed of the base-station, and determines the time for filtering the results in the case of client/server solutions.

F indicates the filtering factor, i.e., the ratio

$$F = D_{filtered}/D_{res} \quad (1)$$

where $D_{filtered}$ indicates the size of the filtered results. Since $D_{filtered} \leq D_{res}$, the value of F is between 0 and 1.

The parameters are summarized in Table 1 for readers' convenience. The total response time can be expressed as the sum of two factors: the time between the formulation of the query and the start of the elaboration of the query in the database site (T_{query}) and the time between the start of the elaboration of the query in the database site and the availability of the filtered results in the base-station (T_{result}).

Single queries In a `client/server` solution, the query is elaborated at the database site, thus T_{query} is given by the time needed to transmit the query to the database, i.e., the network latency times the size of the query: $\tau\ D_{query}$.

T_{result} is given by the time required to extract the results from the database (T_{db}), the time needed to convert the stream of results in a string transmittable over the network (T_{str}), the time to transmit the results to the base-station ($\tau\ D_{res}$) and the time needed to filter the results in the base-station ($BS\ D_{res}$) (preliminary experiments have shown that the time for marshalling the query is negligible, and thus it has not been considered in the model):

$$T_{tot} = [\tau\ D_{query}] + [T_{db} + T_{str} + \tau\ D_{res} + BS\ D_{res}] \quad (2)$$

T_{query} is relevant only in the case of very slow networks (high value of τ). T_{result} is usually the dominant factor, and strongly depends on the size of the produced results, which influence all factors comprising T_{result}. Typical genome queries usually produce large results and the dominant factors become the database response time (T_{db}) and the the time needed to send the results to the client machine ($\tau\ D_{res}$).

DIMENSIONS OF DATA	
D_{query}	Dimension of the query
D_{agent}	Dimension of agent and associated plan
D_{res}	Dimension of the results
ELABORATION AND TRANSMISSION TIME	
T_{db}	Database response time
T_{str}	String conversion time
τ	Network latency
T_{ser}	Agent serialization time
ENVIRONMENTAL FACTORS	
DB	Database processing speed
BS	Base-station processing speed
F	Filtering Factor
RESULTS	
T_{tot}	Total response time
T_{query}	Time to transmit the query to the DB
T_{result}	Time to compute and return the results

Table 1. Summary of the parameters of the analytical models

In a `remote evaluation` solution, the query is elaborated at the database site, as in the case of client/server solutions, but the results are filtered remotely (at the DB site), thus in a remote evaluation solution, both the time to transmit the results to the base-station and the time needed to filter the results are less than the one required in a client/server solution. The term τD_{rev} that substitutes the term τD_{query} includes the extra time required to send the program as well as the query to the remote stations. The transmission time is reduced by the filtering factor F, since data are filtered before being transmitted, while the filtering time depends on the speed of the DB site and not of the base station:[1]

$$T_{tot} = [\tau\ D_{rev}] + [T_{db} + T_{str} + \tau\ D_{res}\ \mathbf{F} + \mathbf{DB}\ D_{res}] \quad (3)$$

Thus, the benefits of remote evaluation over client/server solutions depend on the filtering factor, the network latency and the difference in speed between the base-station and the database site. As confirmed by the case studies reported in the next section, the difference is appreciable only on relatively slow networks, such as ADSL connections.

In a `mobile agent` solution, the base-station sends the whole agent to the database site, and not only the query.

[1] The differences between related formulas are shown in bold in the paper.

The agent is evaluated at the DB site, which sends back the filtered results. Thus, only T_{query} changes with respect to remote evaluation solutions. It now depends on the size of the agent and not only on the size of the query, and includes also the time for serializing the agent:

$$T_{tot} = [\tau\ \mathbf{D_{agent}} + \mathbf{T_{ser}}] + [T_{db} + T_{str} + \tau\ D_{res}\ F + DB\ D_{res}] \quad (4)$$

The benefits of mobile agent solutions with respect to client/server solutions depend on the same factors of remote evaluation solutions, but are reduced by the serialization and transmission time of the agent that is higher than the transmission time of the query.

In the case of simple queries over single databases, the analytical models suggest potentially better performances of remote evaluation over mobile agent solutions, and indicate the filtering factors and the network latency as critical parameters for gaining better performances of mobile over client/server solutions.

Incremental queries on a single database Incremental queries on a single database iteratively access the database to refine the results. Thus, the new analytic models can be obtained from the single query/single database models by accumulating the costs of the iterative steps as follows:

client/server

$$T_{tot} = \sum_{i=1}^{n} \big[[(\tau D_{query,i} + \tau \mathbf{D_{res,i\text{-}1}} \mathbf{F_{i\text{-}1}})] + [(T_{db,i} + T_{str,i} + \tau D_{res,i} + BS\ D_{res,i})]\big] \quad (5)$$

remote evaluation

$$T_{tot} = \sum_{i=1}^{n} \big[[(\tau D_{rev,i} + \tau \mathbf{D_{res,i\text{-}1}} \mathbf{F_{i\text{-}1}})] + T_{db,i} + T_{str,i} + \tau D_{res,i} F_i + DB\ D_{res,i}\big] \quad (6)$$

mobile agent

$$T_{tot} = [\tau D_{agent} + T_{ser}] + T_{str,n} + \sum_{i=1}^{n} \big[T_{db,i} + DB\ D_{res,i}\big] + \tau D_{res,n} F_n \quad (7)$$

In the formulas, the indexes i identify the different iterative accesses, e.g., $D_{query,i}$ indicates the dimension of the query during the i-th iteration. Boldface indicates the differences with the case of the simple query. We notice that for both client/server and remote evaluation solutions, at each iteration the base-station must send to the database not only the i-th version of the query, but also the filtered results of the former query (as indicated by the terms $\tau\ D_{res,i-1}\ F_{i-1}$) to enable the new iteration at the database site. The symmetry of the formulas with respect to the case of the single query suggests no main differences in the relative performances of the two classes of solutions. Conversely, mobile agent solutions accumulate less overhead. In fact, since the agent is evaluated on the database site, we need to send the agent only once to the database, and we do not need to exchange intermediate results for incremental filtering that is performed directly on the database site by the agent. This reflects in the first ($\tau\ D_{agent} + T_{ser} + T_{str,n}$) and last ($\tau\ D_{res,n}\ F_n$) terms of the formula that are not included in the sum. This gives a performance advantage to mobile agent solutions over other solutions. The performance gap grows for queries that require several refinements, as often the case in the considered applicative domain.

Incremental Queries on several databases Incremental queries on several databases iteratively access different databases to refine or enrich the results of a query. The new analytic models are parametric with respect to the number n of queries and the number m of nodes, and can be obtained from the incremental queries/single database models by considering the costs of the iterative steps for each database, as follows:

client/server

$$T_{tot} = \sum_{i=1}^{n} \big[[(\tau D_{query,i} + \tau D_{res,i-1} F_{i-1})] + [(T_{db,i} + T_{str,i} + \tau D_{res,i} + BS\ D_{res,i})]\big] \quad (8)$$

remote evaluation

$$T_{tot} = \sum_{i=1}^{n} \big[[(\tau D_{rev,i} + \tau D_{res,i-1} F_{i-1})] + T_{db,i} + T_{str,i} + \tau D_{res,i} F_i + \mathbf{DB_i}\ D_{res,i}\big] \quad (9)$$

mobile agent

$$T_{tot} = \mathbf{m}(\tau D_{agent} + T_{ser}) + \sum_{\mathbf{j=1}}^{\mathbf{m}} \big[[\tau \mathbf{D_{res,j\text{-}1}} \mathbf{F_{j\text{-}1}}] + \mathbf{T_{str,j}}\big] + \sum_{i=1}^{n} \big[T_{db,i} + DB_i\ D_{res,i}\big] + \tau\ D_{res,n}\ F_n \quad (10)$$

Boldface indicates the differences from the single database case. We notice that the client/server and the remote evaluation models do not change. In fact, client/server solutions return the intermediate results to the client station that filers the data and executes the new query, while remote evaluation solutions filter the intermediate results on the database station, but return the filtered data to the client station to formulate the successive query. The

index in the term DB_i in the remote evaluation model indicates that data are filtered on the database where they are retrieved, which in this case changes at every step.

The mobile agent solution differs from the single database case, since the agent must be sent to the m nodes involved in the search (the term $\tau\ D_{agent} + T_{ser}$ now is computed m times), and the intermediate results must be sent to the new database when the next subquery involves a different database (a term $\tau\ D_{res,j-1}\ F_{j-1} + T_{str,j}$ is added for the m nodes involved in the search.) Thus, the advantages of the mobile agent over the remote evaluation solution are reduced and depend on the transmission time between databases. Fast inter-database connections result in small overhead, while slow connections can result in loos of performances below the performances of remote evaluation solutions.

Independent queries on several databases Independent queries on several databases iteratively access different databases with queries that are non correlated. In a client/server solution the base-station must iteratively elaborate each query, while both mobile solutions can take advantage of the parallel evaluation on the independent databases. Thus, the advantages of the mobile solutions over the client/server solution are emphasized by a factor corresponding to the number of databases interested by the queries.

Summary The analytical models confirm the hypothesis of better response time of mobile over client/server solutions. The advantages increase with the number of databases and the independency of the queries. Although in general, mobile agent solutions seem better than remote evaluation solutions, advantages are not always significative and in few cases remote evaluation seems to perform slightly better than mobile agent.

4 Case Studies

We evaluated the analytical predictions on some case studies using prototypes that support the three solutions: client/server, remote evaluation and mobile agent.

The three prototypes have been implemented in Java and share a common core that interfaces with the databases and elaborates the extracted data to satisfy the users' queries. The remote agent prototype is obtained by adding a service to transfer queries and drivers on the database machines. Drivers execute the code on the machine hosting the database. The mobile agent solution is obtained by extending the agent class with the μCode platform [13].

The prototypes are extended with plug-ins for handling different databases. The current prototypes offer plug-ins for five popular databases: *EMBL* [17], *SwissProt* [19], *PubMed* [21], *InterPro* [18] and *BLAST* [20].

EMBL stores information about nucleotide sequences [11], SwissProt contains information about protein sequences [2], PubMed stores bibliographical references, InterPro contains information about protein families, and BLAST provides facilities for searching for similarities between protein and nucleotide sequences.

Both the remote evaluation and the mobile agent prototypes require the code to be executed on the database machine. Moreover, the mobile agent prototype requires the availability of the μCode platform on the database machine. Most databases do not allow execution of alien code, thus, to conduct our experiments, we installed copies of EMBL (150GB) and SwissProt (6GB) databases on our own machines, and we experimented with these databases.

The prototypes have been instrumented to record the execution time of the relevant phases. All prototypes record the database response time (T_{db}), the string conversion time (T_{str}), the time for elaborating the results, which corresponds to $BS\ D_{res}$ for the client server prototype and to $DB\ D_{res}$ otherwise, and the time for sending the data to the base station, which corresponds to $\tau\ D_{res}$ for the client server prototype and to $\tau\ D_{res}\ F$ otherwise. The mobile agent prototype records also the serialization time (T_{ser}) and the time for transmitting the agent to the remote station ($\tau\ D_{agent}$). Both the remote evaluation and the mobile agent prototypes record also the time spent to send the driver to the remote station, which was not considered in the analytical formulas.

The experiments have been executed with a benchmark identified in collaboration with Schering-Plough. The benchmark consists of sets of queries that represent typical genomic queries in the different cases identified in the previous sections: single queries, incremental queries on a single database and incremental queries on several databases. Each set contains queries that vary in terms of dimensions of the produced results.

Tables 2, 3 and 4 show some sample queries that are referred to in the discussion of the next section. Table 2 shows some simple queries on the SwissProt and the EMBL databases, respectively (databases contain different information and thus are accessed with different queries). Table 3 shows four incremental queries on a single database: SwissProt. Table 4 shows six incremental queries on two databases: SwissProt and EMBL.

5 Empirical Results

We empirically evaluated the different solutions proposed in this paper by running the three prototypes (client/server, remote evaluation, and mobile agent) with the

SWISSPROT

QA1	Output Field: All		Output Field: All Not contain "receptor"		Output Field: ID	
	(Kb)	F	(Kb)	F	(Kb)	F
Field: All "Cannabinoid"	80	1	20	0,25	0,2	0,002

QA2	Output Field: All		Output Field: All Not contain "suin"		Output Field: ID	
	(Kb)	F	(Kb)	F	(Kb)	F
Field: Organism "Pig"	4750	1	1100	0,23	12,5	0,002

QA3	Output Field: All		Output Field: All Not contain "protein"		Output Field: ID	
	(Kb)	F	(Kb)	F	(Kb)	F
Field: Organism "Mouse"	32000	1	3500	0,11	97	0,003

EMBL

QA4	Output Field: All		Output Field: All Contain "nucleic"		Output Field: ID	
	(Kb)	F	(Kb)	F	(Kb)	F
Field: Author "Martinelli"	46	1	23	0,5	1,5	0,032

QA5	Output Field: All		Output Field: All Contain "cds"		Output Field: MedLineID	
	(Kb)	F	(Kb)	F	(Kb)	F
Field: Organism "Golden Hamster"	4610	1	1200	0,26	15,6	0,003

QA6	Output Field: All		Output Field: All Contain "rel.65"		Output Field: ID	
	(Kb)	F	(Kb)	F	(Kb)	F
Field: Organism "Dog Hookworm"	30100	1	4100	0,14	138	0,004

A simple query is characterized by a field and an input string. The database returns the records that contain the string in the given field. The three columns indicate the size of the output both in Kbytes and in percentage with respect to all retrieved records with three different filters. Filters are given in the header of the columns as constraints on the fields. The first column corresponds to no filter, the second column corresponds to a filter that leads to a moderate filtering factor, the third column corresponds to a filter that produces a high reduction on the size of produced data.

Table 2. Sample single queries.

QB1	All "cannabinoid" ⟶ Author ⟶ MedLineID ⟶ Description
QB2	Organism "pig" ⟶ ID ⟶ Author
QB3	Organism "frog" ⟶ ID ⟶ Description
QB4	Organism "mouse" ⟶ ID ⟶ AccessN

An incremental query is characterized by a sequence of simple queries. Each simple query is performed on the set of records produced by the former query. For example, query QB1 is composed of three simple queries. The first query selects the content of field ”Author” of all records containing ”cannabinoid”. The second query selects the content of fields ”MedLineID” of all records containing any of the values returned by the former query as author. In this way it retrieves all products of authors who dealt with ”cannabinoid”. The third and last query selects the content of field ”Description” associated to all records whose field ”MedLineID” matches the results of the former query. In this way we have a description of all work of authors who dealt with ”cannabinoid”. The other queries work similarly.

Table 3. Sample incremental Queries on SwissProt.

QC1	SWISSPROT	EMBL
	Organisme "giraffe" ⟶ MedLineID ⟶ Author	
QC2	EMBL	SWISSPROT
	Organism "duck" ⟶ ProteinID ⟶ InterProID	
QC3	EMBL	SWISSPROT
	Author "martinelli" ⟶ ID ⟶ Protein ID ⟶ Description	
QC4	SWISSPROT	EMBL
	All "cannabinoid" ⟶ ProteinID ⟶ AccessN	
QC5	EMBL	SWISSPROT
	Organism "coffee" ⟶ ProteinID ⟶ InterProID	
QC6	SWISSPROT	EMBL
	Organism "pig" ⟶ MedLineID ⟶ ID	

Table 4. Sample incremental queries on SwissProt and EMBL.

three kinds of queries discussed in the previous section (single query, incremental query on single database and incremental query on several databases) under different bandwidth conditions: LAN (10Mbit/s), WAN (1Mbit/s) and ADSL (256Kbit/s).

All experiments were conducted using local images of the SwissProt and the EMBL databases, to overcome problems of running “alien” code on publicly available databases, as required to execute the remote evaluation and mobile agent solutions. The different bandwidth conditions were obtained using local, remote and ADSL-connected computers to run the client, respectively. The local client was an AMD Atholon XP 1700+ processor with 512MB of RAM. The remote client, located in Eugene Oregon, was an Intel Pentium 4 processor 1.50GHz with 512MB of RAM, and the ADSL-connected computer was an Intel Pentium 4 processor 2.40GHz with 256MB of RAM. The server machines, that store the EMBL and SwissProt databases, were Intel XEON dual processor 2.00GHz with 1GB of RAM.

We ran all experiments with generic drivers pre-installed on the servers running the databases. Thus, the total execution time measured in the experiments does not include the time for transmitting and installing the generic drivers.

The mobile agent prototype serializes the agents before

transmitting them on the network. The total execution time measured in the experiments includes the serialization time, which in our case accounts for a fixed overhead of 20ms.

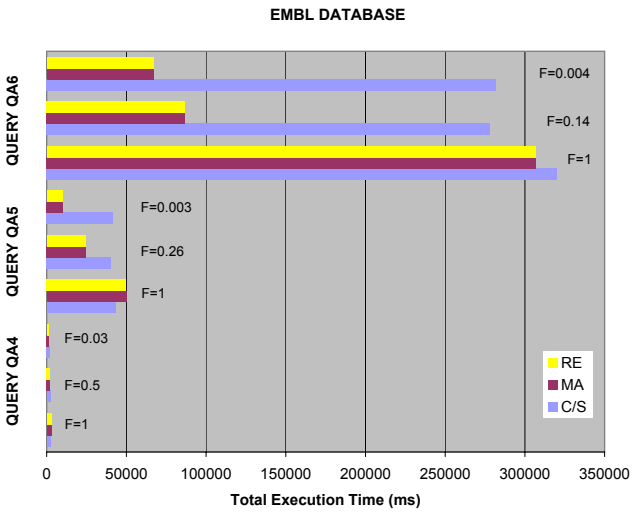
The experiments involved the execution of 2520 queries that derive from the execution of the 16 queries illustrated in the previous section (6 single queries, 4 incremental queries on a single database and 6 incremental queries on multiple databases) with or without filtering (in the case of single queries only) for the three solutions (client/server, remote evaluation, and mobile agent) under the three bandwidth conditions (LAN, WAN, ADSL). Each query has been executed 10 times on different dates and times of day to avoid dependencies from the specific load conditions. We then eliminated the outliers and we considered the mean values.

5.1 Single Queries

In the case of single queries, mobile agent and remote evaluation solutions do not present significant mutual differences, and both usually perform better than the client/server solution. The advantage over the client server solution decreases with the speed of the network. Figure 2 shows the total execution time for the first three queries of Table 2 with three different filtering factors each, executed on EMBL using a WAN connection. The advantages of remote and mobile solutions over client/server depend on the amount of the returned data and the filtering factor. For small quantity of returned data (query $QA4$) or no filtering (Queries $QA5_{F=1}$ and $QA6_{F=1}$), all solutions are equivalent. For medium quantities returned data and no filtering ($QA5_{F=1}$), client/server performs even slightly better than the remote and mobile solutions, due to the additional cost of agent transferring not balanced by a lower cost of returning data. However, for the critical cases of large amounts of returned data and high filtering factors ($QA5_{F=0.26}$, $QA5_{F=0.003}$ and especially $QA6_{F=0.14}$ and $QA6_{F=0.004}$), remote and mobile solutions perform much better than client server solutions. We measured an average gain of a 2.2 factor and a maximum gain of 4.3 factor of mobile over client server solutions.

LAN and ADSL present similar behaviors, but were reduced in the case of LAN, and were enhanced in the case of ADSL. In our experiments, with a LAN connection the gain reached a maximum factor of 2, while with an ADSL connection it reached a factor of approximately 17 (from 20 minutes with the client/server prototype to 1 minute for the mobile agent prototype in the case of query $QA6_{F=0.004}$).

The experiments with SwissProt confirmed the result obtained with EMBL.



Total Execution Time for the different solutions client/server (C/S), remote evaluation (RE) and mobile agent (MA). The three rows of each query correspond to the results for the three filtering factors (F) of Table 2.

Figure 2. Single queries on EMBL on a WAN.

5.2 Incremental queries on a single database

In our experiments with incremental queries on a single database with WAN connectivity, the mobile agent solutions gain up to a factor of 2 over the remote evaluation solutions, which gain a factor of 2 on the client/server solutions. The amount of gain depends on the amount of incremental filtering. The remote evaluation solutions gain over the client/server by returning less data at each step, while the mobile solutions gain by returning only the last result.

Table 5 shows the ratios between the performances of the three solutions for the four queries of Table 3 on SwissProt with a WAN. The differences in the gains between the queries depend on the reductions on the exchanged results due to incremental filtering.

As for single queries, the advantages leverage on high-speed networks (LAN), and are enhanced on low-speed networks (ADSL): The ratio among the different approaches approximates 1 with LAN connectivity and reaches a gain factor of up to 17.2 between mobile agents and client/server solutions with ADSL connectivity.

5.3 Incremental queries on several databases

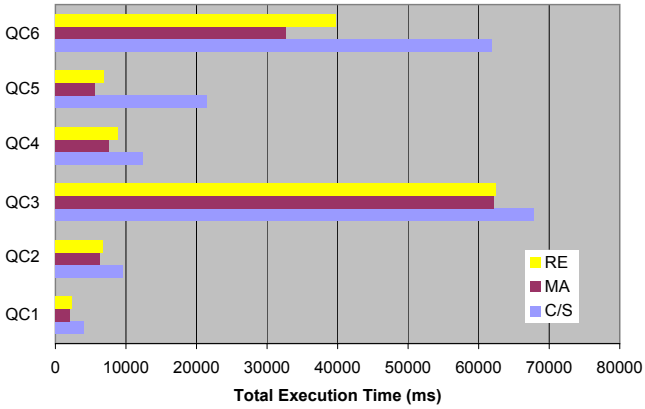
In the cases of incremental queries over several databases the experimental results confirm once more the analytical predictions: mobile solutions outstand client/server approaches and mobile agents perform slightly better than remote evaluation approaches.

	QB1	QB2	QB3	QB4
CS/MA	1.63	3.88	3.37	4.28
CS/RE	1.03	1.97	1.73	2.07
RE/MA	1.58	1.97	1.94	2.07

Ratio between the performances of the different solutions (client/server over mobile agent (CS/MA), client/server over remote evaluation (CS/RE) and remote evaluation over mobile agent (RE/MA)) for queries with different incremental filtering factors.

Table 5. Incremental queries on SwissProt on a WAN.

Figure 3 shows the total execution time for the six queries of Table 4 with WAN connectivity. The experiments have been executed with only two databases (EMBL and SwissProt) working on the same LAN, due to the impossibility of running the experiments on the native databases, and to the problems of autonomously replicating databases. Thus the results are conservative: we should expect higher differences when executing with multiple databases and with WAN interconnections.



Total Execution Time for the different solutions client/server (C/S), remote evaluation (RE) and mobile agent (MA).

Figure 3. Incremental queries on SwissProt and EMBL on a WAN.

6 Conclusions

The amount of data involved in the research on genome makes classic client/server applications inefficient. Federated databases, data-warehousing and text mining mechanisms alleviate but do not solve the problem. In this paper, we evaluate mobile code approaches to solve the problem, we study the benefits with simple analytical models, and we report on a set of case studies that show the benefits of remote evaluation and mobile code solutions over client/server solutions for different classes of queries. The experiments have been executed with prototypes on WAN, LAN and ADSL networks with two popular databases, SwissProt and EMBL, with a set of queries identified with experts of the application domain. The results obtained are very encouraging and we plan to develop a precompetitive tool for beta testing to collect detailed information for complete evaluation of the approach.

7 Acknowledgments

This project started with a collaboration with Schering-Plough. We are grateful to the research group in Milan for the valuable information provided during the project. We would like to express our gratitude especially to Dr. Massimo Beltramo. We are also grateful to Luca Andreini, Jonathan Loroni and Lorenzo Vassallo who contributed in the early stages of the project and to the development of the prototypes.

References

[1] P. Argos and T. Etzold. SRS, an indexing and retrieval tool for flat file data libraries. *Computational applied biosciences*, (9):49–64, 1993.

[2] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1998. *Nucleic Acids Research*, 26(1):38–42, 1998.

[3] M. Baldi and G. P. Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In R. Kemmerer, editor, *Proceedings of the 20th International Conference on Software Engineering*, pages 146–155. IEEE CS Press, Apr. 1998.

[4] D. Chess, C. Harrison, and A. Kershenbaum. Mobile agents: Are they a good idea? research report, IBM Research Division, 1995.

[5] B. Donnelly. Data integration technologies: an unfulfilled revolution in the drug discovery process? In *BIOSILICO, Drug Discovery Today*, pages 59–63. Elsevier Science, May 2003.

[6] R. Feldman, Y. Regev, E. Hurvitz, and M. Finkelstein-Landau. Mining the biomedical literature using semantic analysis and natural language processing techniques. In *BIOSILICO, Drug Discovery Today*, pages 69–80. Elsevier Science, May 2003.

[7] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[8] R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, and D. Rus. D'agents: applications and performance of a mobile-agent system. *Softw. Pract. Exper.*, 32(6):543–573, 2002.

[9] R. Jha and S. Iyer. Performance comparision of implementation mechanisms for e-commerce applications: Towards a hybrid approach. In *Proceedings of the 2nd Asian International Mobile Computing Conference (AMOC'02)*, May 2002.

[10] D. Kotz, G. Jiang, R. Gray, G. Cybenko, and R. A. Peterson. Performance analysis of mobile agents for filtering data streams on wireless networks. In *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 85–94. ACM Press, 2000.

[11] T. Kulikova, P. Aldebert, N. Althorpe, W. Baker, K. Bates, P. Browne, A. van den Broek, G. Cochrane, K. Duggan, R. Eberhardt, N. Faruque, M. Garcia-Pastor, N. Harte, C. Kanz, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, R. Mancuso, M. McHale, F. Nardone, V. Silventoinen, P. Stoehr, G. Stoesser, M. A. Tuli, K. Tzouvara, R. Vaughan, D. Wu, W. Zhu, and R. Apweiler. The EMBL nucleotide sequence database. *Nucleic Acids Research, Database Issue*, 32:D27–D30, 2004.

[12] S. Papastavrou, G. Samaras, and E. Pitoura. Mobile agents for world wide web distributed database access. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):802–820, Sep/Oct 2000.

[13] G. P. Picco. μCODE: A Lightweight and Flexible Mobile Code Toolkit. In *Proceedings of Mobile Agents: 2^{nd} International Workshop MA'98*, volume 1477 of *Lecture Notes on Computer Science*, pages 160–171, Stuttgart (Germany), Sept. 1998. Springer.

[14] G. P. Picco. Mobile Agents: An Introduction. *Journal of Microprocessors and Microsystems*, (25):65–74, 2001. Invited contribution to a special issue on mobile agents.

[15] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM Press, 1998.

[16] C. Sigrist, L. Cerutti, N. Hulo, A. Gattiker, L. Falquet, M. Pagni, A. Bairoch, and P. Bucher. PROSITE: a documented database using patterns and profiles as motif descriptors. *Brief Bioinform*, 3(3):265–274, 2002.

[17] www.ebi.ac.uk/embl/. EMBL Home Page.

[18] www.ebi.ac.uk/interpro/. InterPro Home Page.

[19] www.ebi.ac.uk/swissprot/. SwissProt Home Page.

[20] www.ncbi.nlm.nih.gov/BLAST/. BLAST Home Page.

[21] www.ncbi.nlm.nih.gov/entrez/query.fcgi. PubMed Home Page.