Parallelization of R-programs with GridR in a GPS-trajectory mining application

Dennis Wegener, Dirk Hecker, Christine Körner, Michael May and Michael Mock

Fraunhofer Institute Intelligent Analysis and Information Systems
Schloss Birlinghoven, 53754 St. Augustin, Germany
{dennis.wegener, dirk.hecker, christine.koerner, michael.may, michael.mock}
@iais.fraunhofer.de

Abstract. In this paper, we show how computing intensive R-applications can be parallelized in a grid environment using GridR, our tool for the seamless integration of R-programs in grid environments. We apply the GridR parallelization in a GPS-trajectory mining application that calculates the reach and gross contacts of outdoor poster advertisements and demands for scalability. The application provides the foundation for price calculations for all outdoor advertisement throughout Germany. In our approach to parallelization, users are free to submit sequential or parallel R-programs to the grid or directly to cluster-based parallel execution environments. The execution is mapped transparently to the chosen execution environment in such a way that neither the user nor the R-programmer of the data mining algorithm is affected. We present the overall approach to the parallelization of R-applications and the performance results achieved with the GPS-trajectory mining application.

Keywords: Cluster Computing, R statistical language, parallelization, grid, GridR, GPS, spatio-temporal data mining

1 Introduction

Ubiquitous Knowledge Discovery investigates learning in mobile and distributed environments. To do so, we need to utilize infrastructures that support distributed data mining tasks in a flexible manner. Sensor networks, peer-to-peer networks, grid computing are examples of such infrastructures. In this paper we investigate a highly flexible tool for distributing and parallelizing data mining applications in grid environments using the R language for statistical computing. We investigate its usefulness in a specific application setting: track mining from GPS data; the tool itself however is completely generic.

The analysis of spatio-temporal data is a broad field of research. During the past years, several GPS studies have been conducted in transportation research to analyze the travel behavior of persons. Most of the studies are based on GPS data collected within a single city with up to a few hundred test persons [1, 2]. A rather large GPS

study has been conducted in Switzerland, including about 10,000 test persons from 12 conurbations [3].

The data mining application described in this paper computes the reach and gross contacts of outdoor poster campaigns based on GPS-tracks of a set of test persons. Reach is defined as the percentage of a population exposed to a campaign within a certain period of time (e.g., a week). Gross contacts are the total number of contacts a campaign achieves. Both measures serve as common currency for comparing the performance of campaigns in the advertisement business. Typical users are located in marketing departments of a production company and want to evaluate the reach of poster campaigns for outdoor advertisements. The data on which the evaluation takes place was collected in a nationwide study in Germany and consists of trajectories that reflect the outdoor movements of a representative sample of 30,000 test persons. The data is stored in a central database in our computing cluster, in which the computation is executed.

The considered GPS-trajectory mining application is written in the statistical language R [4]. The algorithm makes use of simulation and advanced statistics and is extremely computing intensive. The scenario used for evaluation comprises 12 German cities and would have taken roughly a year to compute on a conventional PC, what is not acceptable for the application scenario. In addition, taking into account that calculations might be performed nation-wide for each city in the country, scalability is a key issue for this application. In conclusion, the application must be executed in parallel on a high-performance computing cluster, thus raising the demand for a parallelization and a remote access to computing clusters for R-programs. We have developed a grid-based analysis tool called GridR [5], which provides the user of R with a convenient access to distributed computing resources. In this paper, we show how this access can be extended to R-programs that run in parallel either on a computing cluster or on other computing resources made available through the grid. We show techniques to parallelize R-applications with GridR either on the client-side or on the server side. Furthermore, the R-program for parallelization and the R-program of the original data-mining algorithm can be separated, such that the programmer of the data-mining algorithm is not affected. In the considered GPS-trajectory mining application, the execution of the R-application is mapped transparently onto a computing cluster which itself is managed by the Condor system [6]. We present and discuss the performance gain of the parallelization and show that the parallelization of the computation achieves acceptable response times, even if each of the parallel tasks requires access to a central database. In addition, we present the performance results of the poster reach estimation for different poster campaigns.

The remainder of the paper is organized as follows: section 2 presents the application scenario. The GridR tool and its parallelization capabilities are presented in section 3, and section 4 reports on the parallelization of the GPS-trajectory mining application. Related work is presented in section 5. We conclude with a short summary in section 6.

2 The AGMA Application

In Germany, the outdoor advertisement industry records a yearly turnover of more than 800 million euros. The Arbeitsgemeinschaft Media-Analyse e.V. (ag.ma) – a joint industry committee of around 250 principal companies of the advertising and media industry in Germany – authorized the AGMA project, which provides the foundation for price calculations in outdoor advertisement throughout Germany. In 2006/07 the ag.ma commissioned a nationwide survey about mobile behavior and appointed Fraunhofer IAIS to calculate the reach and gross contacts of poster networks [7].

Fig. 1 on the left shows a campaign of 321 billboards on arterial roads in Cologne. The right hand side displays the development of reach over a period of seven days. Reach is a time-dependent measure about the publicity of a poster network. It states the percentage of people which see at least one poster of the campaign within a given period of time, e.g. one week. The campaign reaches about 50 percent of the Cologne population on the first day, after one week about 90 percent of the population have seen a poster of the campaign.

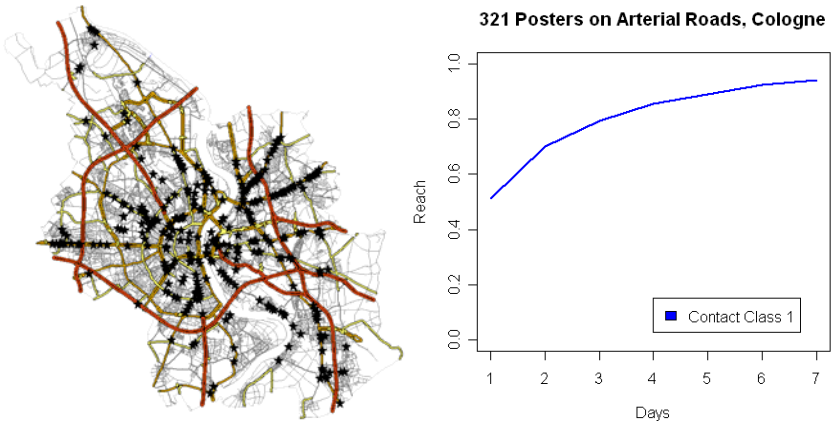

Fig. 1. Reach of a poster network in Cologne

The basic input data of the AGMA application are trajectories and poster information. Nationwide, the daily movements of about 30,000 people have been surveyed using GPS technology and telephone interviews. This data amounts to about 21 million tuples where each tuple represents a section of a trajectory which has been mapped to the street network. Poster information indicating geographic location, type and visibility is available for approximately 230,000 posters. In order to determine reach and gross contacts, the intersections of a given poster network and the trajectory data have to be calculated. However, the number of test persons is rather small compared to the whole population of Germany and the trajectories do not span the full street network. Therefore, we introduce variance to the data by performing a geographically restricted simulation of the trajectories. The resulting simulated trajectories serve as basis for computation of reach and gross contacts. However, the trajectories suffer

from incompleteness in terms of missing measurements due to defective GPS devices or the forgetfulness of test person, which easily interrupt the series of measurement days. These deficiencies are treated in the modelling step by applying the Kaplan-Meier survival analysis technique [8]. Details of the analysis technique itself are not subject of this paper but are under preparation for a separate publication. A detailed documentation of the study is already available at the ag.ma website[1] (in German).

Our computations are implemented using the statistical software R, as R directly supports statistical analysis including the Kaplan-Meier method. At the beginning, the script retrieves input data by triggering several database queries that read a (random) network of posters and the test persons' movement data. Afterwards, a 100-fold simulation of trajectories is performed, and reach and gross contacts are calculated. In previous tests, we determined a number of 100 simulations to achieve stable results. Finally, the results are stored in a text file.

In addition to calculations regarding a particular poster campaign, the advertising industry is also interested in mean network ratings. This requires the repeated execution of the script with randomly selected poster campaigns of specified size, and subsequent averaging of results. Depending on the size of the poster network, stable results can be achieved by using 30-100 repetitions.

The input parameters allow for a variety of combinations. For instance, the geographic location of a poster network can span from a single city up to combinations of cities to federal states or whole Germany. Also, the target population can vary. For instance, it is interesting to know the amount of contacts contributed by people living in the surrounding area of a large city. Other parameters are the size of the campaign and the poster type. Depending on parameterization, the execution time to calculate reach and gross contacts of a single campaign varies between several minutes and a few days. Thereby, the main influence factors are the number of target test persons and the size of the poster network. Clearly, the large number of parameterizations prohibits advance computation of all possible settings, so that computations must be performed on demand. This requires an execution in reasonable time, which applies in particular to mean computations, where 30-100 repetitions of a setting must be executed. Therefore, scalability plays a crucial role in the application.

3 Parallel processing with GridR

3.1 GridR

In previous work, we developed GridR [5] as analysis tool based on the statistical environment R [4] that allows using the collection of methodologies available as R packages in a grid environment. The R language is a de facto standard for statistical research and is used by many applied statistics projects as it provides a broad range of state-of-the-art statistical and graphical techniques as well as advanced data mining methods. In GridR, the R environment has been grid enabled in a way that it allows to be used as user interface for accessing grid resources and invoking grid services. GridR was mainly developed in the context of the ACGT project [9] with the main

[1] http://www.agma-mmc.de/03_forschung/plakat.asp?topnav=10&subnav=199

goal to enable users to gain profit from using a distributed environment (speed-up, scalability, etc.) within R scripts but without the need for dealing with the complexity of the distributed environment.

The GridR toolkit consists of three components: the GridR R environment, the GridR services and the GridR client. The GridR R environment basically is a standard R environment with some additional packages installed. The GridR services are high level components implemented as grid or web services that, e.g. as in the context of ACGT, can be used in a workflow environment, provide additional functionality like connecting to R script repositories, interface with scheduling components built on top of grid middleware, etc. The GridR client, which is implemented in form of an R package, is a very flexible tool which can be used for contacting components responsible for resource management in distributed environments. It can interface with the GridR services or directly with grid-middleware components or cluster management systems and in particular provides the functionality of remote method invocation in a distributed environment. In detail, the GridR client contains components for submitting jobs to the GT4 grid middleware [10], to the Condor system [6] which can also interface with grid middleware as well as to simple pools of machines contacted via SSH.

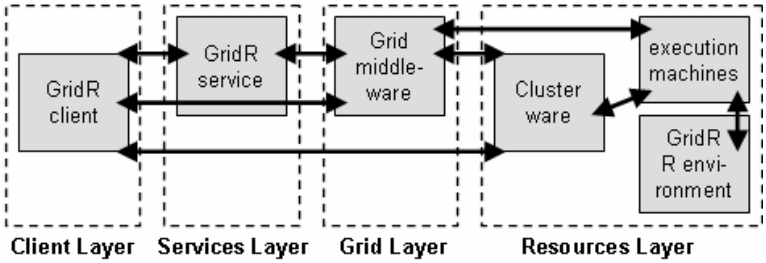

Fig. 2. Layered architecture of a grid system with 4 layers. The components of GridR are depicted to the respective layers. The arrows visualize in which way and across which layers the GridR components are able to interface with the each other as well as other components of the architecture and show the flexibility of the GridR toolkit.

Most of today's large grid-based systems are based upon layered architectures, consisting e.g., of a Client Layer containing components like the user interfaces, a Services Layer providing high level services and interfaces, a Grid Layer containing grid middleware components, and a Resource Layer that contains the hard- and software resources accessible in the environment. Fig. 2 describes a layered architecture as it is, e.g., developed within the ACGT project, in an abstract way and depicts to which layers the GridR components belong.

3.2 Parallelization Technique

In the following, we present a way of how to make use of parallel computation using the GridR client component of the GridR toolkit. Parallelism can be expressed in the workflow at the client layer in GridR. It provides the functionality of submitting R code packaged in a function as a job for execution in a distributed environment. The

jobs are submitted in the background and a resource management system, e.g. the Globus Toolkit or Condor, can process them in parallel. Fig. 3 visualizes the approach of client side parallelization by generating a number of jobs at the client side, submitting them via the appropriate interfaces to a resource management system and processing them on the execution machines.

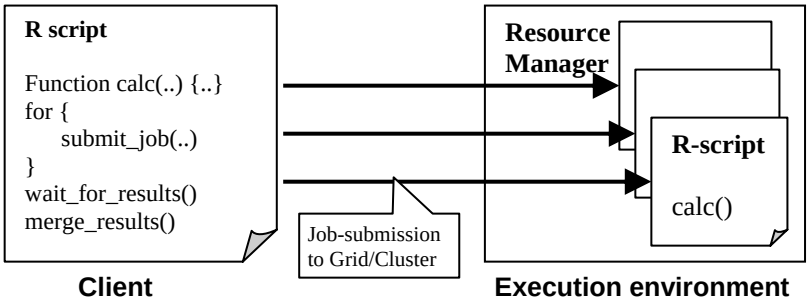

Fig. 3. GridR client side parallelization – On the client side an R script is executed that submits a number of jobs (e.g., inside a for-loop) to resource manager, e.g. the GT4 grid middleware or the Condor cluster management system. The jobs run in parallel in the execution environment and compute the same R function (e.g. with different parameter settings).

We now describe a technique of server side parallelization of R scripts which relieves the clients from handling parallelism. Instead of submitting a number of independent tasks that are computed in parallel, a single job is submitted that is processed as parallel job. As in a sequential execution, the client just has to submit an R script *calc()* containing the algorithm that performs the analysis as a single job to a resource management system, but wrapped by another function *wrap()*. In addition, this function also has a parameter indicating the desired parallelization degree that has to be specified. The script containing the wrapper function which is executed on the server side is responsible for the generation of subtasks as well as the result aggregation. Generation of subtasks means in detail that from the execution process on the server side another job-submission process is started. This can be achieved easily by instantiating the GridR client on the server side. This instance of the GridR client submits a number of sub-jobs to a resource management system which process the actual computation tasks on the execution machines.

With this approach of instantiating the GridR client on the server side, where the server takes the role of a client during the processing of jobs, users are enabled to set up complex parallel applications flexibly by even involving different resource management systems or environments (grid, cluster) as well as to make use of recursion. With this approach it is still possible to modify the R script specifying the algorithm independently from the parallelization because the latter is packaged into a separate wrapper function. It would be even possible to set up a collection of predefined wrapper scripts for some tasks (e.g. result aggregation) which would allow combining wrappers with different algorithms depending on the scenario.

Hence, there are several possibilities of combining different grid- or cluster-based resource managers by also making use of recursion. Fig. 4 visualizes this approach of server side parallelization as an example of submitting a single job from the client

side to the grid and submitting a number of parallel sub-jobs from the grid execution environment to a cluster. This setting is also later used in the presented application scenario in Section 4.

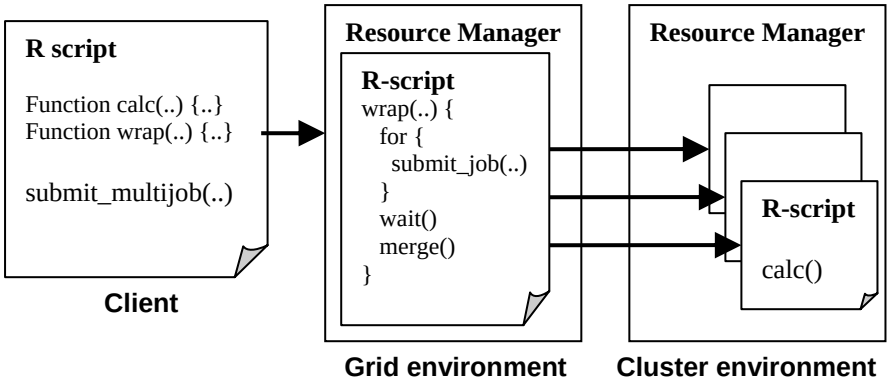

Fig. 4. Example of GridR server side parallelization – Instead of generating a number of jobs at the client side, the client submits just a single job to a resource management system, e.g. the GT4 grid middleware. In the execution environment, the job is processed and computes the wrapper function, which internally starts launching a number of sub-jobs that are computed in parallel to a resource manager, e.g. the Condor cluster management system, again. By this, the functionality for client side parallelization (see Fig. 3) can be used on server side in the execution environment even in a recursive way.

4 Parallelization of the application

4.1 Motivation

The scenario we selected for experimental evaluation calculates mean network ratings for a (rather small) number of poster types and network sizes in 12 cities, amounting to a total of 92 parameterizations. Each parameterization was repeated 100 times for averaging. If computed on a stand-alone machine, the expected runtime of the experiment would amount to about one year. For being able to compute the scenario at all, the application has to be speeded up. Candidate technique for this is the parallelization of subtasks of the application's algorithm. Hence, it has to be clear if the algorithm can be parallelized at all. In our application scenario, the algorithms can be parallelized in theory because it contains a loop with independent steps based on randomly fetched data. But, it is not directly obvious if a parallelization will improve the runtime in practice because the data is stored in a central database which could be a bottleneck.

A further constraint in the application scenario is that the GPS data is confidential, which results in the fact that the database is only accessible from machines in the FhG cluster. For this reason, the application was not deployed in a grid environment but rather in an environment where the computation is performed on a single computing cluster.

Moreover, the algorithm developer/data miner has no knowledge about distributed environments, grid middleware and tools for parallel programming. As the algorithm is written in R, GridR appears as candidate tool for the parallelization of the application as it provides the access to distributed environments in a seamless way.

The following sections give details on how the application was parallelized using the presented technique as well as on the evaluation of the gain of the parallelization in praxis, done by several experiments.

4.2 Parallelization

In this section, the technique of parallelization is applied to the application. The application scenario is as follows. The client, which is used by the marketing experts in the advertising companies, contacts a resource management system of a distributed environment and submits jobs (computing the reach of poster networks) that are processed on the dedicated execution machines. In our case, a computing cluster managed by Condor and the data provided by our institute was used. The Condor batch processing system [6] is a cluster management system for computing-intensive jobs, providing mechanisms for queuing, scheduling and resource management. Jobs can be submitted to the Condor system as independent tasks. Condor places them in a queue and decides upon a policy when and where to run the tasks. Technically, R is the executable that is called when the Condor jobs are processed on the execution machines in the pool, taking the wrapper script, which contains the analysis script, as input.

In our application scenario, the wrapper script submits n subtasks to Condor, each taking the analysis script as input now. These subtasks compute the AGMA scenario in parallel. They draw a random sample of posters along with the trajectory data from a central database and compute the poster reach. The wrapper script waits for the appearance of all results and then averages them. In the AGMA scenario, averaging of n=100 parallel runs varying the poster network was required to achieve the desired stability in the result.

Summarizing, GridR together with the presented technique of parallelization enables users to submit a single job that is automatically split up in a number of parallel tasks on the server side. As an advantage, the code that is executed on the client side becomes smaller as there is no need for result checking and result aggregation on the client side. Furthermore, communication overhead is reduced because the splitting into parallel tasks and the result aggregation are performed on the cluster side. Consequently, performance is increased as computing clusters as, e.g., a Condor pool, are specially set up for allowing high speed communication between the execution machines.

4.3 Experiments

We conducted artificial experiments with a reduced computational load and real-world experiments to evaluate our system. The goal of the artificial experiments was to test whether it is useful to parallelize the application at all, having in mind that each parallel task has to get the trajectory data from the central database, thus inducing a

parallel load on the database server. The big real-world application then delivered the data proving the feasibility of our system by computing the reach of 92 poster campaigns in 12 German cities.

The setup of our experiments was as follows: The experiments were processed on a PC cluster managed by Condor. In total, the cluster consists of 30 AMD Opteron 2.2 GHz machines running Condor on Linux. Each machine holds 2 CPUs, 8 GB memory and two local HDDs with 120 GB. The database machine is an Intel Dual Core 2x2.6 GHz with 4 GB memory, 2x250 GB HDD (1 x System-HDD, 1 x DB-HDD), running Oracle 10.2 on Linux. All machines are connected by a 1GBit network connection. Throughout the experiments it could not be guaranteed that all resources of the pool were free and accessible for the full period of computation.

Artificial experiment. Each of the parallel tasks first draws the trajectory data and randomly chosen posters from the central database server and then locally computes the reach of the poster campaigns. As all parallel tasks compete in the database access, the ratio between the time needed for the database access and local computation has a direct effect on the expected speedup in a parallelization. We conducted experiments in which the size *k* of the inner simulation loop of the local computation was varied from 1 to 100, with 100 being required in the real-world experiment. The results are shown in Table 1.

Table 1. Local execution of a task with varying simulation loop size *k* (in seconds).

Simulation loop size *k* :	1	20	40	60	80	100
Computation time:	143	799	1,496	2,197	2,897	3,569
% database acc.:	75.5%	12.1%	6.5%	4.4%	3%	2.7%

The chosen task computed the reach of a poster campaign with 321 posters, retrieving from the database the trajectories collected by 535 persons over a week in Cologne (about 430,000 street segments, roughly 7 MB). As we can see from Table 1, even if the database access takes 75.5% in the (hypothetical) worst-case of minimal local computation (*k*=1), only 2.7% of the computation time are spent with the access to the database system in the parameter setting required for the real world application (k=100), thus giving a high chance for achieving speedup in parallelization.

Based on this positive result, we conducted an experiment to find out whether the central database server was a bottleneck in the parallel execution. We studied the hypothetical worst case in which 75.5% of the computation time was spent on database access (*k*=1) and varied the number *n* of parallel executions for different poster networks. The results are shown in Table 2:

Table 2. Parallel execution of *n* tasks with 75.5% database access (in seconds).

# parallel tasks *n*:	5	10	20	30	40	50	60
Cluster runtime:	140	150	330	320	450	540	320
Aggregated runtime:	625	1,251	2,502	3,754	5,005	6,257	7,508
Speedup:	4.4	8.3	7.5	11.7	11.7	11.6	23.4

Table 2 depicts the results in the case in which *n* parallel tasks are started simultaneously and actually access the data at the same time. The cluster runtime denotes the job computation time on the cluster, the aggregated runtime is the sum of the execution times of the individual tasks, and the speedup is defined by the ratio between the aggregated and the cluster runtime (we use this definition in order to be consistent with the real-world experiment described below). The computation was parameterized with k=1 to let each local task spend 75.5% of the time with the database access. Given these conditions, the results are promising: the database system did not dramatically slow down the parallel executions. We noted, however that the non-predictable caching effects of the database seemed to have direct effects on the speedup achieved. This is explained by the high fraction of database access in the computation from Table 2. In the real-world application, however, the percentage of database usage is 2.7% only and the parallel runs are much more interleaved. The artificial experiments correspond to the campaign depicted in line 3 of Table 3 below, in which we achieved a much higher speedup.

Real world experiment. We tested the runtime behavior for the computation of average campaign ratings in 12 cities. In each city, poster networks of the type column (C), billboard (BB), city light poster (CLP) or mega light (ML) were drawn respecting different campaign sizes. Each parameterization was averaged over a group of 100 tasks, amounting to 9,200 tasks in total. Table 3 shows an excerpt of the job parameterizations and execution times for the city of Cologne.

As stated earlier, the runtime of a task depends upon the number of test persons available for the city and the size of the campaign, which is derived from the advertising pressure. Thus, the scenario provides a variety of input data. Fig. 5 left shows the average task runtime depending linearly on the volume of input data, which is defined as the number of test persons multiplied by the number of posters.

Table 3. Parameterization and results showing runtimes (in seconds) and speedup for the computation of values for the city of Cologne – 8 jobs with 100 tasks.

Poster type	Advertising pressure	# Poster	# Test persons	Average run-time per task	Aggregated runtime	Cluster runtime	Speed-up
CLP	normal	740	533	8,326	832,566	21,488	39
BB	high	483	533	4,619	461,939	20,104	23
BB	medium	321	533	3,482	348,169	8,315	42
BB	low	241	533	2,832	283,218	7,751	37
C	high	161	533	3,353	335,288	12,069	28
C	medium	121	533	2,751	275,071	9,092	30
C	low	97	533	2,362	236,245	10,641	22
ML	normal	102	533	1,712	171,173	8,243	21

Fig. 5 right displays the relationship between the sum of individual task runtimes and the total runtime on the cluster for all 92 jobs. On average, we obtained a speedup of 45 which is a plausible result for the execution of 100 tasks on a cluster of 60 machines. On evaluation of the results, we detected four anomalous jobs (marked with triangles in Fig. 5 right), which did not terminate properly because some of their

tasks failed. The extreme outlier on the top left results most likely from external jobs, which competed for cluster and database resources.

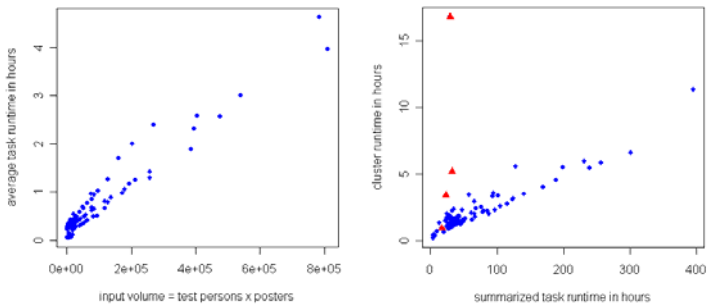

Fig. 5. left: Runtime behavior vs. input volume; right: Cluster runtime vs. sum of individual job runtime

5 Related Work

In recent years, a number of research projects developed middleware packages that enable users to access remote computing clusters. Grid technology such as the popular GT4 toolkit [10] makes computational resources available across geographical and organizational borders. European projects such as ACGT [9] establish an integrated grid infrastructure for a pan-European analysis of medical cancer data. While remote computing clusters made available by grid technology are mostly used for the submission of independent computations, parallel computations can either be defined in the workflow at the client layer or have to be implemented explicitly by the use of parallel programming libraries such as MPICH-G2 [11] or Ibis for Java [12]. Both solutions are not transparent to the user. As a consequence, in most cases, computing clusters made accessible by grid technology are not considered as “parallel machines”. The same holds for cluster management systems such as Condor [6] that allow starting independent jobs in a cluster and then take care of load balancing, accounting, etc. A lot of components and tools, e.g. like MPICH-G2 as environment for parallel programming, Condor as cluster management system or DataMiningGrid [13] as grid-environment aim at providing support for computing different kinds of programs in distributed environments. However, these efforts lack of integration with the R environment on the client side as they support only the execution of R scripts taking R as executable in the respective execution environment.

In [14] an overview on the state-of-the-art research in the context of R and web and grid services is given. Some rudimentary support for building client-server applications that use R on the server side has been offered by toolkits like Rserve [15] or Rweb [16]. Support for concurrent computations in R is provided by packages such as rpvm [17], rmpi [18] and snow (Simple Network Of Workstations) [19]. Rpmv and rmpi provide wrappers to the parallel programming packages parallel virtual machine (PVM) [20] and message-passing interface (MPI) [21] respectively. These approaches

require explicit orchestration of message passing in the parallel execution of R scripts and are only suitable for closely-coupled homogeneous environments. The snow package provides a higher level of abstraction that is independent of the communication technology. pR [22] supports a fully transparent and automatic parallelization of R code based on MPI. However, in contrast to GridR, pR as well as the other efforts discussed in this paragraph do not offer a seamless integration with grid technology.

6 Conclusion and outlook

In this paper, we have shown and applied techniques for the parallelization of R-programs in grid environments using our grid-tool GridR. The considered application enables marketing departments from all over Germany to compute the reach of outdoor poster campaigns based on trajectories of test persons. As the R-based computation of the reach requires substantial computational efforts, the application requires access to a pool of execution machines that executes the analysis tasks and holds the data. With GridR, we enable the parallel execution of R scripts on resource management systems in distributed environments, e.g. on the grid as well as on clusters. Using this technique in our application, a client can submit and initiate parallel computations of R scripts. In the considered application, we managed to compute the complete scenario of 12 cities in a few days compared to the hypothetical sequential execution time of roughly a year.

In addition to the case study presented, the ability introduced by GridR to bind the R-execution environment to distributed infrastructures opens a large field of further potential use cases that arise from the wide use and the richness of the R-language. R itself already provides a lot of interfaces to different data sources, e.g. databases, data-streams or file systems. Hence, scenarios are possible that, in contrast to the considered application, are based on distributed data. Moreover, GridR already supports partially (for some resource managers) or can be extended to support sending jobs to dedicated machines for execution so that users can specify which (part of the) algorithm is to be processed on which machine. This feature can be used to set up distributed R-programs that process continuous data.

Although the application described in this paper features spatio-temporal data collected by mobile devices and utilizes distributed mining, it is only a first step towards ubiquitous knowledge discovery. Independently of the current project we are developing a mobile application that replaces standard GPS-devices by mobile phones [23]. This GPS tracking application for mobile phones subsumes GPS tracking and GPS track annotation and will allow for a distributed data collection exploiting a web-based client/server design that supports storing the (annotated) GPS data directly on a remote database system.

Acknowledgements. The authors gratefully acknowledge the support of the ACGT project that is funded by the European Commission (FP6/2004/IST-026996).

References

1. Bhat, C.R., Srinivasan, S., Bricka, S.: Conversion of Volunteer-collected GPS Diary Data into Travel Time Performance Measures: Literature Review, Data Requirements, and Data Acquisition Efforts, Research Report 5176-1, Center for Transportation Research, The University of Texas at Austin (2004)
2. Bricka, S.: Non-Response Challenges in GPS-based Surveys, Resource Paper Prepared for the May 2008 International Steering Committee on Travel Survey Conferences Workshop on Non-Response Challenges in GPS-based Surveys (2008)
3. Martial, P., Hofmann, U., Mende, F.H., May, M., Hecker D., Körner, C.: Modelling and prospects of the audience measurement for outdoor advertising based on data collection using GPS devices (electronic passive measurement system), 8th International Conference on Survey in Transport (2008)
4. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, (2005)
5. Wegener, D., Sengstag, T., Sfakianakis, S., Rüping, S., Assi, A.: GridR: An R-based grid-enabled tool for data analysis in ACGT clinico-genomic trials. In: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, Bangalore, India (2007)
6. Litzkow, M., Livny, M.: Experience with the condor distributed batch system. In: Proc IEEE Workshop on Experimental Distributed Systems (1990)
7. Arbeitsgemeinschaft Media-Analyse e.V.: Infos für Presse, Radio, TV und Online, Press Release 17.12.2007. http://www.agma-mmc.de
8. Kragh, A.P., Ornulf, B., Richard, G.D., Niels, K.: Statistical Models Based on Counting Processes. Springer Series in Statistics. Springer, Berlin Heidelberg New York (1993)
9. The ACGT project (EU): http://eu-acgt.org/
10. Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp. 2-13 (2005)
11. Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed 63(5), pp. 551-563 (2003)
12. van Nieuwpoort, R. V., Maassen, J., Wrzesinska, G., Hofman, R., Jacobs, C., Kielmann, T., Bal, H. E.: Ibis: a flexible and efficient Java-based Grid programming environment. Concurrency & Computation: Practice & Experience 17(7-8), pp. 1079-1107 (2005)
13. Stankovski, V., Swain, M., Kravtsov, V., Niessen, T., Wegener, D, Kindermann, J, Dubitzky, W. Grid-enabling data mining applications with DataMiningGrid: An architectural perspective. Future Generation Computer Systems 24 (4), pp. 259-279 (2008)
14. Li, N., Morgan, M.T., Falcon, S., Gentleman, R.: Approaches to R as web and analytic service. RWebServices Documentation, Fred Hutchinson Cancer Research Center (2006)
15. Urbanek, S.: Rserve - A Fast Way to Provide R Functionality to Applications. In: Proc. of the 3rd International Workshop on Distributed Statistical Computing. Eds.: Kurt Hornik, Friedrich Leisch & Achim Zeileis (2003)
16. Banfield, J.: Rweb: Web-based Statistical Analysis. Journal of Statistical Software, 4(1) (1999)
17. rpvm: R interface to PVM: http://cran.r-project.org/src/contrib/Descriptions/rpvm.html
18. H. Yu. Rmpi package for R. http://www.stats.uwo.ca/faculty/yu/Rmpi/
19. A. Rossini, L. Tierney, and N. Li. Simple parallel statistical computing. UW Biostatistics working paper series (2003)

20. PVM: http://www.csm.ornl.gov/pvm/pvm_home.html
21. MPI Forum: http://www.mpi-forum.org
22. Ma, X. and Li, J. and Samatova, N.F., “Automatic Parallelization of Scripting Languages: Toward Transparent Desktop Parallel Computing”, IEEE International Parallel and Distributed Processing Symposium (2007)
23. Mock, M., Rohs, M.: A GPS Tracking Application with a Tilt- and Motion-Sensing Interface, 2nd Workshop on Mobile and Embedded Interactive Systems (MEIS'08) on the GI-Informatik 2008 Conference, Munich (2008) (accepted for publication)