Development of Data-Intensive Services with Everest

© Oleg Sukhoroslov © Alexander Afanasiev
Institute for Information Transmission Problems of the Russian Academy of Sciences, Moscow
sukhoroslov@iitp.ru afanasiev@iitp.ru

Abstract. The paper considers development of domain-specific web services for processing of large volumes of data on high-performance computing resources. The development of these services is associated with a number of challenges, such as integration with external data repositories, implementation of efficient data transfer, management of user data stored on the resource, execution of data processing jobs and provision of remote access to the data. An approach for building big data processing services on the base of Everest platform is presented. The proposed approach takes into account the characteristic features and supports rapid deployment of these services on the base of existing computing infrastructure. An example of service for short-read sequence alignment that processes the next-generation sequencing data on a Hadoop cluster is described.

Keywords: big data, web services, data transfer, data management, distributed data processing

1 Introduction

The explosive growth of data, observed in a variety of areas from research to commerce, requires the use of high-performance resources and efficient means for storing and processing large amounts of data. During the last decade, the distributed data processing technologies like Hadoop and Spark are emerged. However, the complexity of the hardware and software infrastructure prevents its direct use by non-specialists, and requires the creation of user-friendly tools to solve particular classes of problems.

One way of implementing such tools is the creation of domain-specific services based on the Software as a Service model. This model allows users of such services to quickly, without installing software, reuse ready-made implementations of data processing methods in a particular domain. At the same time, the user does not need to delve into the peculiarities of storing and processing data on high-performance resources behind these services.

Data-intensive services (DIS), in comparison to conventional computational services with a small amount of data, started to develop recently, so the principles and variants of implementation of these services are poorly understood. There are several academic projects aimed at supporting specific areas of research, for example, the Globus Genomics [1] service for analyzing the next-generation sequencing data and the PDACS portal [2] for storing and analyzing data in the cosmology domain. The first system uses Amazon cloud resources as a computing infrastructure, while the second uses the resources of the NERSC and Magellan science cloud. Commercial cloud solutions, such as Amazon ML, Microsoft Azure ML, Databricks Cloud, are general-purpose platforms, including a set of universal services, as well as its own infrastructure for storing and processing data.

There is a lack of best practices for implementation of DIS on the basis of the existing infrastructure for big data processing such as a cluster running Hadoop or Spark platforms which are increasingly used for the analysis of scientific data [3-5]. Also, little attention is paid to the integration of DIS with existing repositories and data warehouses, including the cloud-based ones, as well as other services. A lot of experience in the integration of distributed resources for storing and processing data has been accumulated within the grid infrastructures [6], however these environments are complex for use by researchers and do not support the use of new models of computations and technologies such as Hadoop. Finally, there is a lack of platforms for implementation and deployment of DIS that would provide ready-made solutions of typical problems encountered when creating this kind of services.

This work is designed to fill these gaps. Chapter 2 describes the characteristics and requirements for DIS. Chapter 3 discusses the principles of implementation of the DIS based on the Everest platform, initially focused on creating services working with a small amount of data. A distinctive feature of the proposed approach is support for the rapid implementation of DIS based on available computing resources and data warehouses. Chapter 4 describes an example of a service based on the presented approach for mapping short reads on the Hadoop cluster.

2 Characteristics and Requirements for DIS

Consider typical requirements for DIS that represent remotely available services for solving a certain class of

Proceedings of the XIX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2017), Moscow, Russia, October 10-13, 2017

problems with a large amount of input data. Such services should provide remote interfaces, usually in the form of a web user interface and application programming interface (API). The interface must allow the user to specify the input datasets and parameters of the problem being solved in terms of subject area.

DIS must use high-performance and scalable (normally distributed) implementations of data analysis algorithms, requiring appropriate computing infrastructure for data processing and storage. Such infrastructure is generally represented by one or more computing clusters running Hadoop platform or a similar technology. DIS must translate the user request into one or more computing jobs that are submitted on a cluster and use scalable implementations (e.g., based on MapReduce) of perspective algorithms.

The user must be able to pass arbitrary input data to DIS. If the data is initially located on the user's computer or external storage resource (e.g., a data repository) DIS must implement the transfer of data over a network to the used cluster. When transferring large amounts of data it is important to ensure the maximum transfer rate and automatic failover. Since the process of working with big data is often exploratory, requiring multiple invocations of DIS, the service should support reuse of data loaded to the cluster. In order to optimize the use of network DIS must also cache frequently used datasets on the cluster. Data transfer functions can also be implemented as separate auxiliary services.

Importantly, DIS may operate separately from computing resources used for real data processing. DIS can use multiple resources, that can be situated at different locations. It is also possible that the service uses the resources provided by the user. In such cases it is important for reasons of efficiency to avoid passing the input data from the user to the resource through the service and to transmit the data directly.

In practice, the data analysis is often a multi-step process that requires performing different tasks at different stages of the analysis. In such cases, the results produced by one DIS can be passed as the input to another service. If these services use different resources, there also arises a problem of data transmission between resources. In general DIS should allow the user to download the output to his computer or an external resource, as well as to transfer the data directly to another service. In addition, DIS may provide additional functionality for remote data preview and visualization. These functions may also be implemented as separate auxiliary services.

DIS must support the simultaneous use by multiple users. This requires the protection of user data, resource distribution between users and isolation of computational processes. In the case of cloud infrastructure, DIS must also manage dynamic allocation and deallocation of resources in the cloud, according to the current load.

3 Implementation of DIS with Everest

Everest [7-8] is a web-based distributed computing platform. It provides users with tools to quickly publish and share computing applications as services. The platform also manages execution of applications on external computing resources attached by users. In contrast to traditional distributed computing platforms, Everest implements the Platform as a Service (PaaS) model by providing its functionality via remote web and programming interfaces. A single instance of the platform can be accessed by many users in order to create, run and share applications with each other. The platform implements integration with servers and computing clusters using an agent that runs on the resource side and plays the role of mediator between the platform and resources. The platform is publicly available online to interested users [8].

The advantage of using Everest platform to create DIS is the availability of ready-made tools for rapid deployment of computational services and integration with computing resources that do not require a separate installation of the platform. At the same time, since the platform was originally created to support services with a small amount of data, the effective implementation of DIS on the base of Everest requires a number of improvements. In particular, it is necessary to implement support of direct data transfers from external storage to the resource and vice versa, bypassing the platform. In addition, it is required to implement the integration of the agent with the components of Hadoop platform platform or similar technology used for data storage and processing on the cluster.

Figure 1 presents the proposed scheme of implementation of DIS on the base of Everest platform and existing Hadoop cluster. Consider the scenario of using the service, which includes the following steps marked in the figure.

In step 1, the user uploads the data of interest to some available on the network or selects data already present in the storage. This storage can be represented by cloud services (Dropbox, Google Drive, etc.), scientific data repositories (Dataverse, FigShare, Zenodo, etc.), specialized databases (for example, 1000 Genomes Project), grid services or file servers (HTTP, FTP, GridFTP, rsync protocols). A wide range of existing storage facilities makes the task of integrating DIS with them more important, in comparison with the duplication of their functionality in the service itself. Note that the user's computer can also act as a data store. In this case, the user needs to deploy a software that provides network access to the user's files. The experience of implementing such software to ensure the reliable transfer of scientific data across the network is already available [9].

In step 2, the user prepares and sends a request to the DIS, including a link to the input data and the values of other input parameters required by the service. The passed link should allow downloading the data from the

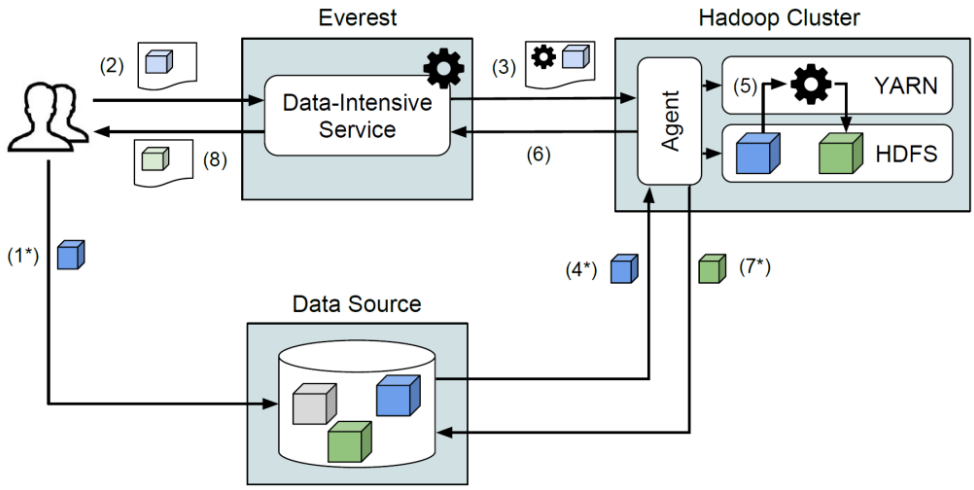

Figure 1 Implementation of DIS on the base of Everest platform and Hadoop cluster

external storage without the user's participation. In some cases, this requires that the user first supply the service with access credentials to the storage, such as an OAuth token or a proxy certificate.

In step 3, based on the user request, the service generates a computational task and sends it to the agent located on the resource used by the service. Together with the task description, the service sends to an agent a link to the input data. As shown in the figure, when sending a task from the service to the agent, the code of the software implementation used for data processing can also be transferred.

The Hadoop and Spark platforms, most commonly used for distributed data processing, use the Java, Scala, and Python languages for implementation of data processing algorithms. Unlike C and Fortran, often used in scientific parallel applications, programs in these languages can be relatively easily transferred from one cluster to another, including their dependencies, without the need for compilation. This opens the possibility for implementation of services on the basis of already created programs and libraries for Hadoop and Spark, which can be used in conjunction with an arbitrary cluster specified by the user. This model significantly simplifies the publication and reuse of developments in this field, without requiring the owner of the service to provide their own resources. This also avoids the multiple implementations of services that use a single program with different resources.

In step 4, the agent downloads input data from the external storage to the local cluster. To implement this step, it is planned to add support for loading data from major types of repositories and storage. Currently the basic support for downloading files via HTTP and FTP protocols, as well as an experimental integration with Dropbox and Dataverse repository are implemented. The downloaded data is placed in the Hadoop Distributed File System (HDFS) on the cluster, where it can be accessed by the program launched in the next step.

In step 5, the agent runs the program specified in the task description on the given input data. The launch is performed through the cluster resource manager such as Yet Another Resource Negotiator (YARN), a component of the Hadoop platform that supports the launch of MapReduce and Spark programs. A special adapter was implemented in order to support interaction of Everest agent with YARN, similar in function to the previously created adapters for integration with HPC batch schedulers.. After the launch, the agent monitors the status of the corresponding job (the application in terms of YARN) and broadcasts the progress information to the service (step 6), which in turn displays this information to the user through the web interface. Upon completion of the program, the agent transmits to the service the output files (of small size) and the final status of the job.

If a large amount of data is produced as a result of the program execution, the agent must support direct network transfer of this data to the user specified external storage (step 7). The information required for this must be transmitted by the user when sending a request to the service in step 2. At the moment, the upload of output data to the specified FTP server or Dropbox folder is implemented.

In step 8, when the request is processed, the service sends the results to the user as a set of output parameters and links to the output files. Some of these files can be stored by the service itself (for example, the program execution log), and some of them can be stored on a cluster or located in an external storage.

Note that the steps 1, 4 and 7, marked with an asterisk and associated with the transmission of data over the network, are not always required or may be omitted. For example, step 1 is not required if the data is already in an external storage or on a cluster, which is true for frequently used data sets. Step 4 can be skipped if the data has already been downloaded to the cluster by the agent or manually by the administrator. To do this, the agent must store information about the downloaded data and cache it for reuse. Step 7 is not required if the received data is an intermediate result

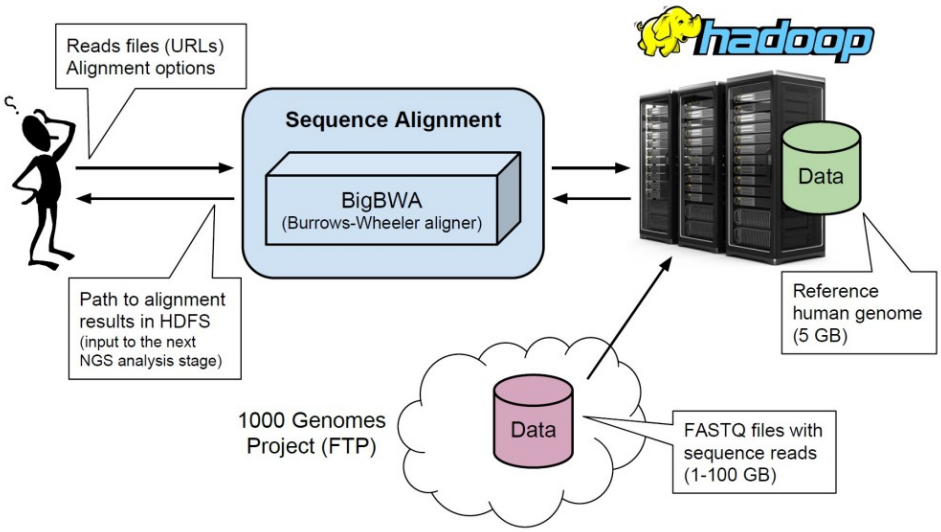

Figure 2 Implementation of DIS for mapping short readings

and will be submitted as an input to another service using the same cluster. Taking into account these cases can significantly reduce the amount of data transferred across the network and, thus, speedup the processing of requests.

Let us briefly consider security issues. Since the service users can not modify the code of the program launched by the service on the cluster, the risk of unauthorized access to data of other users is minimized. When implementing data caching on a cluster, the agent must also limit the re-use of confidential data only by the user who originally provided this data. As for the distribution of cluster resources between users and the isolation of computing processes, these functions are already implemented in the YARN manager.

Although the approach described in this section implies the use of the Hadoop platform, it can be easily adapted to any other big data storage and processing platform.

4 Example DIS Implementation

To demonstrate the described approach, a prototype service was implemented on Everest platform for mapping short readings, one of the basic problems of analyzing the results of the next generation sequencing (NGS) in the bioinformatics domain. This task usually represents the initial and the most computationally intensive stage of the NGS data analysis pipeline, characterized by large volumes of input and output data. The basic scheme of the service implementation in presented in Figure 2.

The service requires one or two (paired) files with reads in the FASTQ format to be provided by a user. The size of these files in compressed form is usually several gigabytes. The public repository of the 1000 Genomes Project was chosen as the main input data storage. This repository provides the ability to download data from the dedicated FTP server where both short reads and reference genomes necessary for solving the mapping problem are available. Therefore the files are provided to the service as links to this or any other FTP server.

To solve the mapping problem, the BigBWA tool [10] was used, which implements the parallel execution of the well-known BWA package (Burrows-Wheeler aligner) in the MapReduce paradigm on the Hadoop cluster. When accessing the service, the user can select one of the mapping algorithms implemented in the BWA package. Additional fine-tuning of the algorithm parameters is currently not implemented. Also, all launches use a fixed reference human genome of about 5 GB in size preloaded on the cluster. The total amount of input data of the problem on test runs was about 10-15 GB.

Upon the request submission, in accordance with the scheme described in Chapter 3, the direct downloading of the read files from the FTP server to the Hadoop cluster takes place. After downloading, the files are uncompresses and converted to the format used by BigBWA. The downloaded files are cached and, if the file link in the request matches the already downloaded one, the data loading step is skipped. After the data is loaded, the MapReduce job is launched with the BigBWA tool.

At the end of the job execution, the service returns to the user the path to the file with the mapping results on the cluster. This approach was chosen because, as noted earlier, reads mapping is only the initial step in the analysis of NGS data. Therefore, in practice, these results will usually be immediately passed as an input to another service in the data processing pipeline. At the user's request, the mapping results can also be uploaded to an external FTP server. The output data on the test runs was about 5-10 GB in the SAM format. In the future, it is planned to convert the results into a more compact BAM format.

The solution of the reads mapping problem on the

Hadoop cluster via the created service allowed to significantly reduce the data processing time. For example, the launch of the BWA package for mapping on two reads on a single server in 4 threads took more than an hour, while the similar launch through a service (28 map-tasks) took about 10 minutes.

5 Conclusion

In this paper, we considered the characteristic features and requirements for the implementation of data-intensive services for working with large data sets. An approach to the implementation of these services based on the Everest platform, initially focused on the creation of computing services with a small amount of data, is proposed. A distinctive feature of this approach, in comparison with commercial cloud solutions, is support for the rapid implementation of services based on existing computing resources and data repositories. An example of a created service that implements the analysis of next-generation sequencing data on the Hadoop cluster is described.

Besides further development of the individual elements of the described approach, future work will focus on remaining challenges. For instance, many existing data repositories are not well prepared for immediate use and require considerable information integration efforts. There is also an increasing demand for processing of data streams. We plan to investigate the use of data integration and stream processing frameworks within the proposed approach to address these issues. We also plan to evaluate the proposed approach on case study applications using larger data sets or combining data from multiple repositories.

Acknowledgements

This work is supported by the Russian Science Foundation (project No. 16-11-10352).

References

[1] Madduri R. et al. Experiences Building Globus Genomics: A Next-generation Sequencing Analysis Service Using Galaxy, Globus, and Amazon Web Services // Concurrency and Computation: Practice and Experience. 2014. Vol. 26, No. 13. P. 2266–2279.

[2] Madduri R. et al. PDACS: A Portal for Data Analysis Services for Cosmological Simulations // Computing in Science & Engineering. 2015. Vol. 17, No 5. P. 18–26.

[3] Ekanayake J., Pallickara S., Fox G. MapReduce for Data Intensive Scientific Analyses // 2008 IEEE International Conference on eScience (eScience'08). IEEE, 2008. P. 277–284.

[4] Zhang Z. et al. Scientific Computing Meets Big Data Technology: An Astronomy Use Case // 2015 IEEE International Conference on Big Data. IEEE, 2015. P. 918–927.

[5] Nothaft F. A. et al. Rethinking Data-intensive Science Using Scalable Analytics Systems // 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015. P. 631–646.

[6] Foster I., Kesselman C. (ed.). The Grid 2: Blueprint for a New Computing Infrastructure. Elsevier, 2003.

[7] Sukhoroslov O., Volkov S., Afanasiev A. A Web-based Platform for Publication and Distributed Execution of Computing Applications // 14th International Symposium on Parallel and Distributed Computing (ISPDC). 2015. P. 175–184.

[8] Everest. http://everest.distcomp.org/

[9] Chard K., Tuecke S., Foster I. Efficient and Secure Transfer, Synchronization, and Sharing of Big Data // Cloud Computing. IEEE, 2014. Vol. 1, No. 3. P. 46–55.

[10] Abu´ın J. M. et al. BigBWA: Approaching the Burrows–Wheeler Aligner to Big Data Technologies // Bioinformatics. 2015. doi:10.1093/bioinformatics/btv506