УДК 004.75, 004.272.2, 004.514, 004.632, 004.624, 608.2

R.A. Rodriges Zalipynis
Donetsk National Technical University
rodriges@csm.donntu.edu.ua

ChronosServer: real-time access to "native" multi-terabyte retrospective data warehouse by thousands of concurrent clients

ChronosServer runs on a cluster of commodity hardware and possess scalability, high availability, and fault tolerance properties. It turns vast amounts of already existing data into actionable intelligence with no changes to the source data files. ChronosServer discovers files on cluster nodes, analyses their structure, and provides format independent SQL-like query model to access their contents. It is capable to read compressed data directly in various formats, including NetCDF, GeoTIFF, GRIB, HDF and many others. This entirely preserves original file metadata as is, vital for its correct interpretation and processing by other software. New data added to the system in a seamless plug-and-play fashion by simple copying it to a cluster node reducing administration overheads. This allows existing software like GIS or statistical packages to operate on files in use by ChronosServer as well as unmodified legacy code to generate data for it. ChronosServer preserves operational infrastructure intact avoiding painful, time-consuming and error-prone data conversions while offering additional opportunities for data analysis.

Data warehouse, real-time access, file formats, metadata, intact operational infrastructure, legacy code

Introduction

The modern world experiences explosive growth of data volumes generated at enormous rates. Many organizations have accumulated and continue collect data in diverse formats. There is software capable to analyze separate data files. However, there remain challenges, including interactive visualization, requiring real-time data access.

The need for the approach is demonstrated on the most largely used data for climate research – reanalysis archives [1, 2, 3, 4, 5]. Their applications vary from deriving simple warming trends to cyclone tracking [7, 8]. These data provide unprecedented opportunities to better understand and, hence, prepare and adapt to storm events, droughts, severe weather conditions and future climate.

The first reanalysis called NCEP/NCAR was released in 1996. Since then, over 9000 publications using its data cite the original paper [9] (6 Nov 2011). During the past decade, more recent reanalysis emerged with higher resolutions and improved models [4, 5].

To obtain the data, researchers repeatedly perform the same set of time-consuming operations by manually selecting and downloading required files. Moreover, they duplicate efforts of their colleagues and encounter the same difficulties due to the absence of intuitive climate data share and visualization tool.

Even primitive climate data visualization is labor-intensive, requires knowledge of format details, distracts from the primary goals and delays results.

Visualization makes the greatest contribution to data understanding. It explores the broader bandwidth of information opposed to text and numbers. With effective visualization, outliers and patterns are easier to perceive what leads to new insights and improved decisions [10].

Surprisingly, in spite of climate reanalysis importance and popularity, tools to interactively explore and visualize them still do not exist.

The main challenge behind this goal is to keep vast amounts of available data on-line for all users while providing real-time access to only a small portion of it for a single client.

Climate Wikience is the first system in the world enabling interactive 3D visualization and analyzes of all available climate reanalysis data in real-time. It is implemented as cloud service [11, 12].

ChronosServer is the backend of Climate Wikience, responsible for the real-time data access by thousands of concurrent clients. By real-time is considered data retrieval without perceivable delay to a human exploring it visually.

Climate Wikience partially implements the Wikience Concept. Nowadays, no single research team is able to explore all available data to it. One of the main Wikience goals is to allow the widest possible audience to do data science effectively and intuitively in order to obtain more value from it.

Currently no straightforward solution exists. Although available systems have powerful distributed models, even the most appropriate of them, HBase [12], solves at most a half of the problem due to difficulties of dealing with sophisticated file formats. At the best, HBase may be used to store a copy of existing data rather than fully switching to it as a storage layer. ChronosServer offers an additional representation and access tier for existing data while not altering their native formats.

Reanalysis Archives

During the history of meteorological observations, data were collected by different instruments and have different character like point measurements at ground stations and areal observations by a satellite. In addition, they are irregularly distributed in time and space, stored in different formats, contain uncertainties, errors and gaps. It is extremely difficult to use that data to obtain a single picture of an atmospheric state [14, 15].

The purpose of a reanalysis archive is to derive the most precise retrospective picture of atmospheric states using all available data and provide the result as time series of regular latitude-longitude grids to simplify its usage. The time step between successive grids and grid resolution are fixed during the whole period. Each node of the grid is assigned a value of a meteorological variable (fig 1).

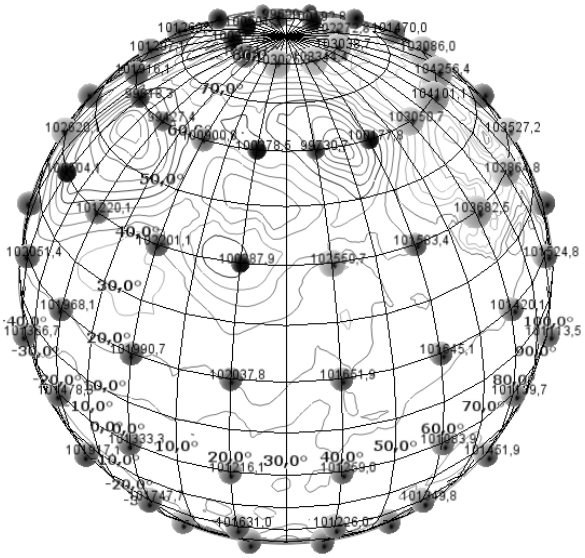

Figure 1 – Mean sea level pressure regular grid with isobars for 01/01/2003 00:00 UTC shown at 20° and 500 GPa intervals respectively. Circle sizes are proportional to pressure values. Built in 3D with Climate Wikience using AMIP/DOE Reanalysis 2

For example, AMIP/DOE Reanalysis 2 (R2) spans several decades, from 01 January 1979 to current date with 6 hour interval and contains over 80 variables. The grid resolution is 2.5° × 2.5° [16].

Climate reanalysis data are usually shipped in NetCDF [17] or GRIB [18] formats. The grids for each variable are stored in a sequence of separate files partitioned by time. File names are prefixed by the code name of the variable.

For example, each file in R2 stores yearly data for one given variable. Thus, surface pressure is stored in files named pres.sfc.1979.nc, pres.sfc.1980.nc, …, pres.sfc.2011.nc. Where "pres.sfc" denotes "surface pressure", 1979 is the year, and ".nc" is file extension for NetCDF format.

Data inside a file is compressed. NetCDF uses simple compression technique. All grid values within a given file are subtracted the constant called "add offset" and divided to "scale factor". To use the values from NetCDF file, they must be multiplied by scale factor and summed with add offset.

Besides the data, NetCDF files store metadata. It includes actual data range, min and max values, constants used for missing values and other important characteristics. This information is easily computed from original data. However, there are datasets where metadata is rich and specific and, thus, vital for correct data interpretation. For example, satellite imagery may have metadata containing radiometer calibration information. Hence, it is preferably to preserve metadata within a storage system as is.

During visual data exploration, a researcher is usually interested in a single grid at a time. A storage system must have the ability to extract only one grid.

ChronosServer Architecture

A ChronosServer cluster consists of multiple workers responsible for data storage and a single gate with Internet connection. Clients interact with ChronosServer through the gate via Internet channel. Workers do not have direct Internet access (fig. 2).

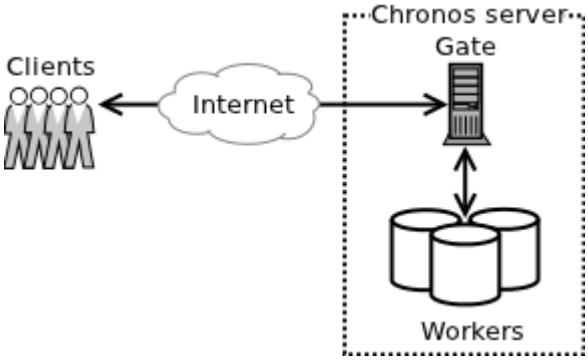

Figure 2 – ChronosServer architecture

A single gate is used to coordinate data access, manage data partitions and transfer data from workers to clients. Although it may become a bottleneck in data exchange, this design was chosen due to constraints with available network addresses. However, it is possible to have several gates if more network addresses are available.

ChronosServer (Chronos is Greek word for time) partitions retrospective data across cluster nodes by time attribute. A series of measurements taken during a period of time with some interval is common nowadays to many domains (recall R2 files that already partitioned by year).

For the sake of clarity, later descriptions will rely on R2 data organization for examples.

The dataset is a storage unit inside ChronosServer representing the complete available period for an R2 variable. A dataset may consist of

arbitrary number of partitions (files). Each partition stores data for a continuous time period. The volume of a single partition is not fixed.

Partitions are replicated across several workers for fault-tolerance. Not preconfigured number of workers may contain the same partition (is called here as variable replication rate). This may be useful for newly arrived data which popularity is high for the first time and decays further.

Datasets are organized in a hierarchy of directories (fig. 3). All workers and the gate have the same hierarchy of data directory on their local filesystems. Thus, a dataset is identified by an unambiguous pathname global to the whole cluster. A worker stores only a subset of all dataset partitions. Each worker stores only a portion the whole ChronosServer directory namespace relevant to the data it possesses. Usually directories are named straightforward following the aliases of reanalysis variables.

The gate keeps on its local hard drive the complete hierarchy of directories for ChronosServer. Instead of storing partitions, its directories may optionally contain additional dataset descriptions, for example, their full names (fig. 3). It may also store rules for data processing before/after extraction or sending the result like unpacking compressed data.

Workers are responsible for reading requested data from a file for a given time snapshot. They do not know what datasets are stored in files on their local hard drives. The gate is not aware of total number of workers in the cluster and total number of partitions for a dataset.

The system is designed as fault tolerant and does not require all workers to be permanently up. Only the master must be a fault-protected machine to maintain the overall service availability. By having a single coordinating machine, sophisticated algorithms for load balancing and data placement can be performed. Thus, it is worth investing into one steady node to benefit from better service.

The gate is unaware of partition locations until a worker itself reports them to it. This is done for better scalability and fault tolerance (fig. 3).

Upon startup workers scan their local filesystem to find out what partitions and time periods do they hold (1). Workers connect to the gate and transmit the list of partitions they store (2). At this point, the gate keeps this information in worker pool – in-memory data structure that maps partitions to their corresponding owners.

On a successful connection, the gate responses to a worker by dataset ids to further reduce network traffic as will be explained in the next section and rules for dataset files processing (3).

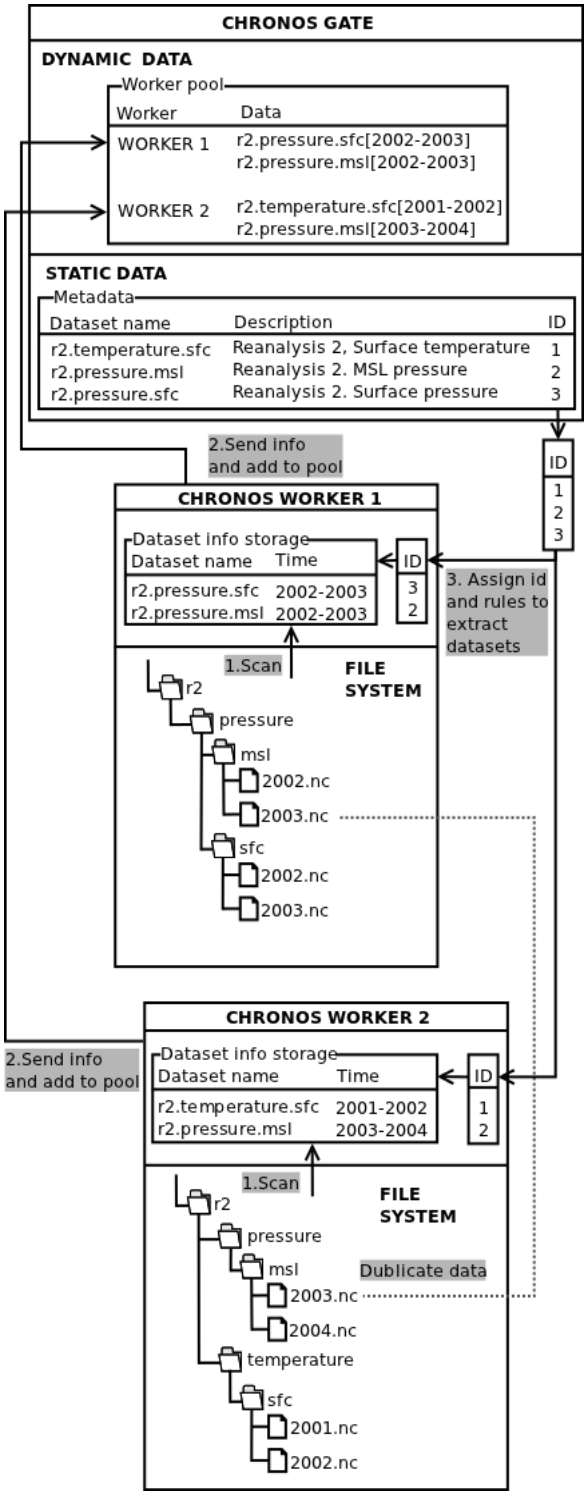

Figure 3 – ChronosServer structure and worker initialization phase

ChronosServer Query Execution

ChronosServer is optimized for small (comparable to dataset size) concurrent random reads in real-time by thousands of clients.

The purpose of ChronosServer is to enable real-time access to data stored in various formats without their modification. It does not aim to build a complete storage infrastructure to rely on in all needs. For example, Climate Wikience uses document-oriented database to store cyclone tracks while at the same time ChronosServer enables it to manipulate with reanalysis archives.

A client interacts with ChronosServer in a query-response fashion. The queries are strings with syntax similar to SQL. This is done for simplicity since the clients are primarily GUI front ends with direct interaction with humans, so that they could clearly see and interact with the processes if needed.

Once a GUI client application – Climate Wikience – starts up, it connects to the ChronosServer cluster through the gate. After the connection is established, it retrieves a list of available datasets by issuing corresponding queries.

During user activity (zoom, pan, viewing time or layer change), Climate Wikience automatically determines required datasets.

The Client – Server interaction is outlined on figure 4. Once time or spatial area is changed, GUI tries to locate data for a dataset in local disk cache (1). Since a usual research pattern is investigating the same period and geographical area during a relatively long time, it makes sense to cache the data in use.

On cache miss, GUI constructs query string with required dataset and time for ChronosServer and transfers the query (2).

A query looks like "SELECT DATA FROM r2.pressure.msl WHERE TIME = 01.02.2003 18:00". It will return R2 regular 2.5°×2.5° latitude-longitude grid values for mean sea level pressure for 2003 Feb 01, 18:00 UTC. Note, that "DATA" and "TIME" are both reserved literals.

The gate receives and parses the query (3). Once it locates workers with the required dataset partition it selects the first of them and shuffles the worker list for better load balance. Gate adds clients to the pool (5) to refer it later and selects dataset id (6) in order to exchange with integers instead of strings to minimize network traffic.

The gate sends the query information to the selected worker (7). It in turn converts id to dataset path (8), finds the partition with given time and directly reads data from the compressed file (9).

Notice, in addition to earlier observations, that a data format usually exploits data peculiarities to compress it efficiently. Reading compressed files directly significantly reduces data storage costs.

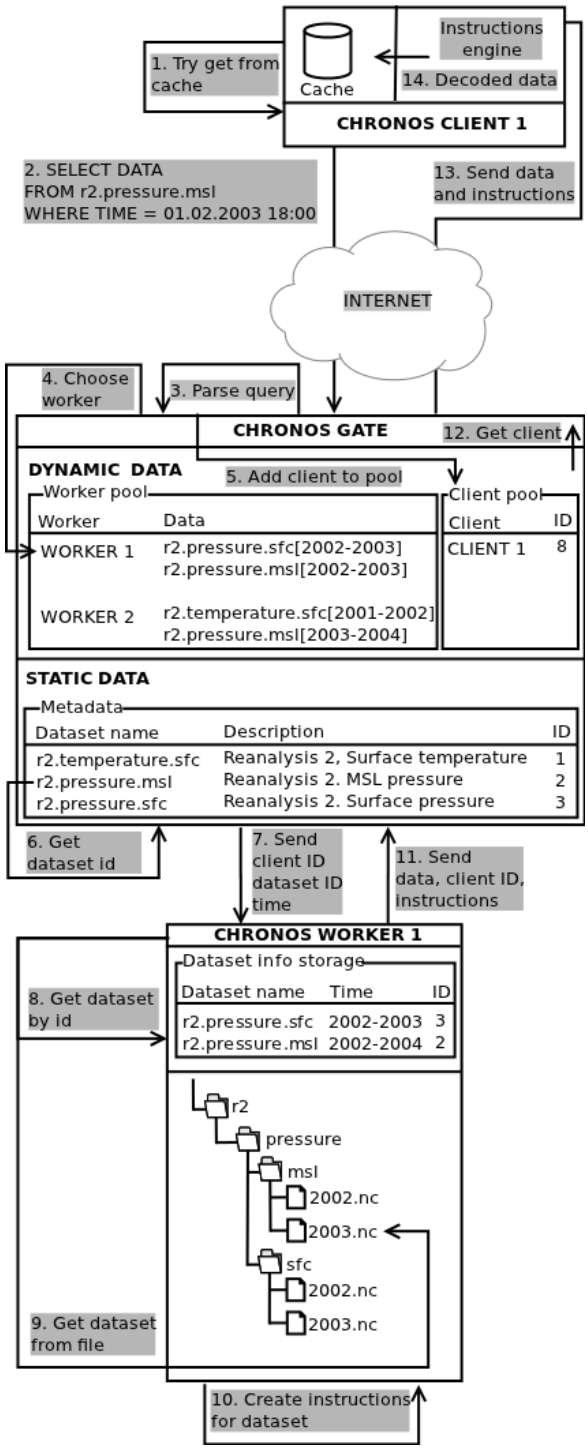

Figure 4 – Client query processing

This paper also introduces "divided query execution" technique to reduce ChronosServer load. Only a portion of required operations are performed on server to obtain ready to use data. Since most scientific file formats store compressed data, they require some preprocessing to unpack them after they are read from file. The proposed architecture exploits the opportunity of having desktop client application.

Instead of applying scale offset and add factor to retrieved data, a worker creates a script (instructions) which client has to run in order to unpack the data. It sends the instructions along with scale factor and add offset values to the client through the gate (11–13).

The client instructions engine interprets the script generated by the worker (14). For small portions of data these additional operations take a very small amount of time (especially on a powerful worker machine). However, for large number of users this may become a serious issue. On the contrary, it is not perceivable by a human since it takes only several milliseconds to execute the script on client side.

Workers do not cache data since the whole available time period of datasets is explored randomly by users what means that all available data are in use and cannot be cached neither in operating memory nor on a gate hard drive.

ChronosServer Performance Evaluation

Worker, gate and client were implemented on Java. Gate and Worker are multithreaded applications and can scale to large number of clients.

The computer cluster was populated by intact files of R2. They are already partitioned by year.

The performance evaluation was carried out using two setups.

The first one has 1 worker, 1 gate and 1, 2, 4, 32, and 128 concurrent clients. Both the gate and the worker have one thread.

The second setup uses 32 threads at gate handling client requests and 32 threads to communicate with workers. Two workers running with 8 threads both.

Machine characteristics are presented in table 1. Workers use extent-based file system ext4fs. It seeks very fast inside a file continuously located on hard drive.

Table 1. Machine characteristics

	OS	RAM	Processor	Java ver.
Gate	Cent OS 6.0	8 GB	Core i7, 8 cores, 3.46 Ghz	1.6.0.27
Worker	Ubuntu 10.04	1 GB	Intel Core Quad 2 2.66GHz	1.6.0.20
Client	Ubuntu 10.10	2 GB	AMD Athlon II Dual-Core P320 (2.1 GHz)	1.6.0.26

Workers are running on VM Ware 7.1 virtual machines configured to have 1 GB of RAM and 2 processor cores.

The configurations of the virtual machines are default. Nothing was tuned. Disk schedulers were not used and even atime attribute was not removed from the file system configuration.

The client and gate use Oracle JRE while the workers use OpenJDK.

Gate, worker and client are connected via a gigabit Ethernet network switch. The client machine is a commodity notebook. It has 100 Mbit Ethernet card while gate and worker have 1 Gbit built-in Ethernet cards.

The ping statistics for sending 10 times by 20Kbytes of data between the worker and the gate has zero packet loss. The round trip time min/avg/max/mdev are 1.318/1.385/1.437/0.047 ms correspondingly. The same ping operation for client-gate communication yields min/avg/max/mdev of round trip time equal to 3.629/3.649/3.676/0.049 ms. Tests were performed when the network was idle.

A multithreaded application on the notebook in turn starts 1, 2, 4, 32, or 128 thread bundles simulating concurrent access of many clients. Each of them connects to the gate on startup.

Each client thread waits the startup of the others. After that, one minute is granted for all threads in total during which they continuously generate queries. A thread randomly chooses a dataset among all available and random time within it. It generates string query requesting data for a single time moment of the chosen dataset (regular grid in case of reanalysis data). A thread does not issue a new query unless it receives a response from the server and applies all required instructions that come along with data. All random number generators were using uniform distribution.

The full query execution time (later simply query time) was measured which includes time from generating query string for the gate to obtaining unpacked data (table 2).

Two kinds of statistics were collected.

For each thread in a bundle, minimum, maximum, average, median query execution time were measured together with the number of queries that a thread had time to execute during a minute given to all threads (not a minute per each thread). Due to space constraints, for each thread bundle only summary statistics is outlined.

The median was calculated in all cases since the longest queries are usually yielded by access overheads to a particular partition for the first time. To avoid it further, workers cache descriptors for opened files. In general, a query is executed much faster than an average.

The time to read a desired data slice from NetCDF file is small due to the extent-based filesystem.

For setup 1 (table 2) and 2 (setup 2) the values are almost the same for each thread within a bundle (± 2 ms), except for 128-threads bundle. Thus, there was no need to take an average, values for a random thread from a bundle are shown.

Table 2. Stages required to process a query and corresponding components responsible for them

#	Stage	Where
1	Create query	Client
2	Pack query	Client
3	Transfer query: Client → Gate	N/A
4	Unpack query	Gate
5	Parse query	Gate
6	Choose worker	Gate
7	Create request	Gate
8	Pack request	Gate
9	Transfer req.: Gate → Worker	N/A
10	Unpack request	Worker
11	Extract data	Worker
12	Create response	Worker
13	Pack response	Worker
14	Transfer resp.: Worker → Gate	N/A
15	Unpack response from worker	Gate
16	Pack response for client	Gate
17	Transfer resp.: Gate → Client	N/A
18	Unpack response	Client
19	Accept response	Client

However, 128-threads bundle exhausts available CPU resources. Most time is wasted for a thread queuing for its time quantum to run, neither performing operations nor even on waiting for data to arrive. Thus, for 128-threads bundle tables 3 and 5 contain average numbers.

Table 3. Statistics for thread bundles (setup 1)

#	Min	Max	Avg.	Med.	Queries
1	7	46	9	9	6101
2	7	102	12	12	4756
4	90	467	160	153	379
32	90	467	160	153	379
128	274	1577	637	568	95

Table 4. Statistics for thread bundles (setup 2)

#	Min	Max	Avg.	Med.	Queries
1	3	34	7	9	7792
2	3	21	9	9	6487
4	3	30	14	13	4232
32	60	162	107	106	560
128	150	1302	434	377	139

To get an idea of situation taking place for 128-threads bundle, figures 5 and 6 present dot charts. The first one plots medium query execution time for setup 1 (upper boxes) and setup 2 (lower triangles) for each thread in 128-thread bundle.

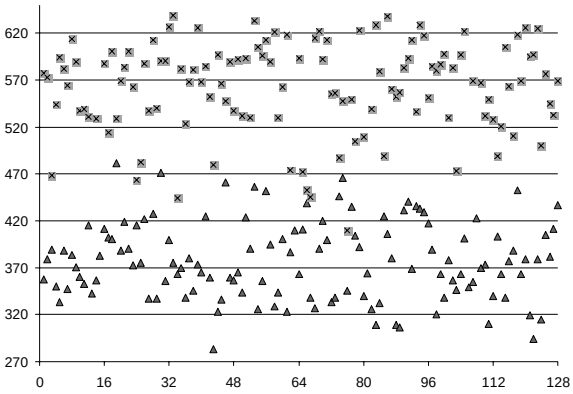

Figure 5 – Medium query execution time for 128 threads from setups 1 and 2

The second one plots the number of queries for each of 128 threads which they managed to perform in a minute time frame for setup 1 (lower boxes) and setup 2 (upper circles).

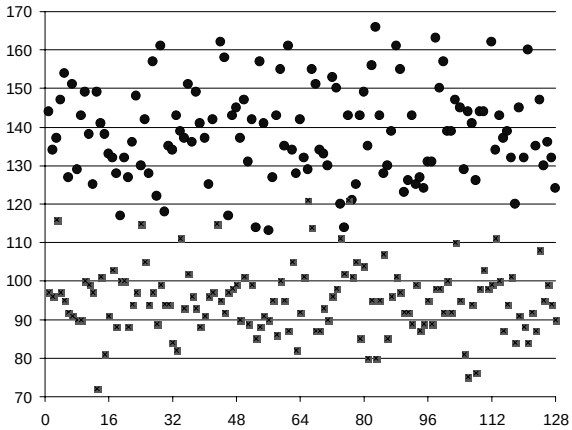

Figure 6 – Number of queries performed by128 threads from setups 1 and 2

Although the number of serving threads at the gate does influence the overall performance, considerable spread can be seen on both plots. Thus, for 128 threads the results may not be representative due to the bottleneck on client side. At the same time, ChronosServer was only slightly loaded.

In case of real world production, the requests will come from diverse network addresses and clients will not be unnaturally overloaded.

The second type of runtime information is the detailed execution time per stage collected on all system components, the gate, worker and client for a single query. Six (6) distinct queries for each setup were selected (tables 5 and 6). The first three are measured for bundle with one thread and the second for randomly chosen thread out of 128-thread bundle. The first query was the fastest, the second was the worst, while the third represents a typical case (median) of all queries for a selected thread.

Table 5. Query execution time per stage, ms (setup 1)

Stage #	1 client			128 clients		
	min	max	med	min	max	med
1	0.1	0.5	0.1	13	48	22
2	0.1	0.5	0.3	16	55	27
3	0.6	2.0	0.7	8	116	20
4	0.2	2.0	0.3	11	43	18
5	0.4	2.0	0.6	13	110	24
6	0.2	1.0	0.4	23	89	27
7	0.2	1.5	0.2	6	43	15
8	0.1	2.0	0.2	10	67	31
9	0.1	5.0	0.3	12	88	20
10	0.2	2.0	0.5	14	43	35
11	1.3	4.5	1.5	15	99	21
12	0.3	2.5	0.3	49	57	50
13	0.4	2.0	0.6	27	107	27
14	0.2	3.5	0.3	13	86	26
15	0.4	5.0	0.5	26	105	45
16	0.4	2.0	0.4	35	112	54
17	0.5	5.0	0.5	16	127	48
18	1.0	1.5	1.1	37	88	45
19	0.3	1.5	0.2	11	40	28
Total	**7**	**46**	**9**	**355**	**1523**	**583**

Presented values justify the fact that in a 128-thread bundle the most of time a thread is just not able to run at all. In table 5, actual time (Act.) was obtained by summing up all values for each stage separately. Measured time (Meas.) was calculated by the thread itself from the beginning of query (generating string) to its completion (obtaining data). For a typical query (median) this difference reaches 277 ms (363–86 ms).

Pie charts on figures 7, 8 and 9 form an idea of how much time it costs for the gate, a worker and a client thread to perform a typical query. Median values were used for all plots.

For setup 2, the gate speed has risen in 10,7 times (214/20) compared to setup 1 (recall that 32 client threads were used for gate in setup 2 and only 1 thread for setup 1). For a single client thread the query execution time has also dropped. This is caused by using 2 workers in setup 2.

Workers evenly share the portion of disk I/O required for query execution.

Approximately 20 Kbytes per query are transferred (the size of a single R2 grid).

Table 6. Query execution time per stage, ms (setup 2)

Stage #	1 client			128 clients		
	min	max	med	min	max	med
1	0.1	0.2	0.2	2.0	2	2
2	0.1	0.2	0.1	1.5	3	2
3	0.2	0.8	0.2	2.0	7	2
4	0.1	0.2	0.2	2.0	3	2
5	0.2	2.3	0.3	0.5	6	2
6	0.1	0.4	0.1	0.5	3	1
7	0.1	0.3	0.1	0.5	5	1
8	0.1	0.4	0.2	1.0	3	2
9	0.1	0.2	0.2	0.5	3	2
10	0.1	0.4	0.2	1.0	3	3
11	0.3	3.6	1.3	4.5	16	8
12	0.1	0.2	0.2	0.5	2	1
13	0.1	0.7	0.1	1.5	5	2
14	0.1	1.2	0.3	1.0	5	1
15	0.3	3.1	0.6	5.0	45	10
16	0.2	0.5	0.2	2.0	3	2
17	0.3	3.3	1.4	3.5	14	10
18	0.3	3.8	2.3	4.0	81	25
19	0.1	1.2	0.8	0.5	59	8
Act.	**3**	**23**	**9**	**34**	**268**	**86**
Meas.	**3**	**23**	**9**	**162**	**1217**	**363**

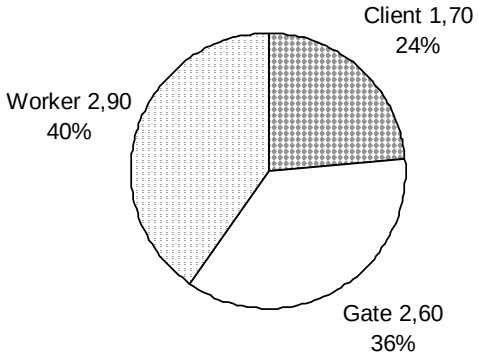

Figure 7 – Setup 1, 1 thread on client

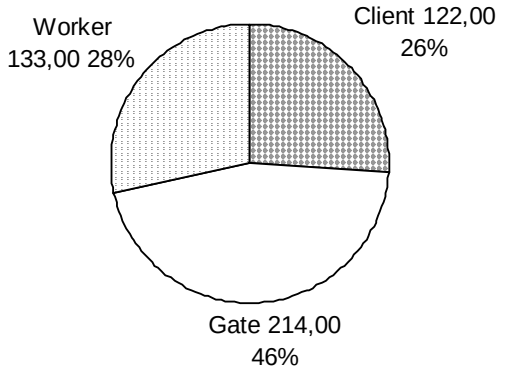

Figure 8 – Setup 1, 128 client threads

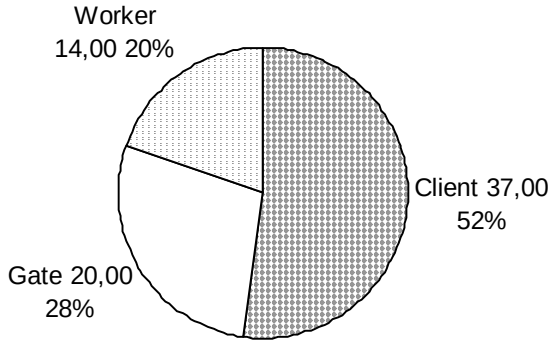

Figure 9 – Setup 2, 128 clients threads

According to median value for 1 thread (table 6) it is achieved 17,36 MBits/sec (1000*20/9/1024*8) transfer rate from ChronosServer to client (query transfer rate) compared to the maximum available network bandwidth of 100 MBits/sec. It transfers 0.16MBits (20/1024*8) in 9 ms while the limit is 0.90MBits in 9 ms (100/1000*9).

It is not completely correct to compare query transfer rate and the maximum network bandwidth. Before a client receives a response from gate, data are read from disk by worker and go through the gate which in turn retransmits the data to the client duplicating network I/O.

However, the overall results show that response times are suitable for real-time serving of thousands of concurrent users. This is justified by two key assumptions.

First, the data is delivered to human users. Once they obtain its visualization, they usually study it via interactive GUI. This takes for a human at least several seconds or even minutes if not days for some cases. Thus, some time is expected to pass after a GUI will issue next query. In the synthetic experimental setups, clients were generating queries uninterruptedly on after another without any delays.

Second, a researcher repeatedly explores the same region or time interval to obtain results. Thus, frequently used data quickly become cached on client side. Physical capabilities of a researcher will allow him/her to embrace during a day only the portion of data that can mostly reside on local hard drive.

Related Work

The techniques for storing large data on a computer cluster may be categorized into distributed file systems, clusters of relational database management systems (RDBMS) and document-oriented databases.

All current systems require the data to be partitioned (in spite of the presence of their natural partitioning), extracted from native formats, and imported into new storage model. Metadata must be maintained separately in a transformed way.

Google File System [19] was designed to run on thousands of commodity machines with scalability and fault-tolerance in mind. It stores files broken on chunks which are distributed among chunkservers. It exploits the single master storing in-memory chunk locations reported by chunkservers on startup.

However, ChronosServer has little in common with GFS since it has directly opposite goals. GFS was designed for batch processing of large data volumes rather than executing thousands of small real-time tasks. It suffers from high latency of data access [20].

HBase [13] is open-source implementation of Google Big Table [21] and is likely the most appropriate solution to the Climate Wikience problem. Because reanalysis data is never modified by a client, the proposed solution is much simpler and less resource demanding. In addition, it can easily incorporate data processing capabilities specific to research needs closer to data and does not require format convertions.

Hadoop [22] with its HBase and other powerful systems is a rich ecosystem ranging from storage to data processing frameworks to cluster monitoring tools. However, many organizations already have vast amounts of accumulated data, human experience and tools with their specific data processing tasks. In addition, they have many other licensed third party software working with files storing their data in diverse formats. Switching to a new infrastructure may be painful, error-prone and require considerable personnel efforts and time.

ChronosServer is designed to offer additional functionality to existing infrastructure. Certainly, a large body of tasks is waiting for HBase in many currently operational infrastructures. It is up to the personnel to decide which system to use depending on the goals they are trying to accomplish.

The RNCEP package [23] for open-source R-statistics system [24] is not designed to store large amounts of data. However, it enables users to select date and variable name for NCEP Reanalysis 1 or 2 in R command prompt. It automatically downloads reanalysis files via Internet from NCEP servers via OpenDAP protocol. All 30MB yearly file is retrieved even if only one snapshot is asked and, certainly, not in real-time. User has to remove unnecessary grids as well as issue other commands to view data in 2D.

Lustre [25] is a very powerful fully POSIX-compliant parallel file system. It runs on a considerable number of clusters from the Top 500 list [26]. It supports RAIDs, parallel I/O and tunes the operating system up to the kernel modification. It partitions large files across cluster nodes.

However, ChronosServer does not aim to implement a file system nor Climate Wikience needs Lustre functionality either. Lustre yields large administration overheads once deployed on a cluster. Notice, however, that it would anyway require a software that reads diverse file formats, thus, Lustre would be not only complicated, but also an incomplete solution.

Document oriented databases offer facilities to store large binary objects inside the database. They also offer sharding and replication. For example, MongoDB [27] uses GridFS [28]. It splits large objects into small fixed-sized chunks, usually 256k in size. Each chunk is stored as a separate document.

Again, one need to invent facilities to preserve metadata and import data into MongoDB from their native formats. In addition, breaking large data into small chunks degrades the performance and introduces storage penalties. Increasing chunk size will make impossible reading a small portion of data from it.

Memory-based distributed systems [29, 30] are not applicable to Climate Wikience problem since the whole available data is expected to be intensively used. This prohibits its in-memory caching.

Parallel databases have real-time response times [31] and some of them have very good scalability [32]. However, database systems are inherently designed for effective operations on small data values. It is much more preferable to keep large data files outside a database [33].

Another approach is to represent the data as raster and use BLOB (Binary Large Object) type to store them inside an RDBMS.

Rasdaman (raster data manager) [34] is the extension to PostgreSQL which partitions a given raster on tiles and stores them as BLOBs. It implements all raster operations itself together with spatial indexing. There is a commercial project [35] aiming to scale Rasdaman. This requires installing PostgreSQL on each cluster node.

However, scaling Rasdaman will be not a trivial task. First, they will have to solve the problem of efficient management of dozens of autonomous PostgreSQL instances. Second, load balancing is hard to implement since it will require individual tiles (BLOBs) residing inside RDBMS to be moved between cluster nodes. The problem is complicated due to the fact that PostgreSQL does not have any scaling tools in contrast even to MySQL [31].

Another major drawback of this approach is that it is impossible to retrieve a portion of a BLOB. Moreover, in PostgreSQL (and, possibly other databases either) it is not possible to store large objects as a whole in a single row. RDBMS breaks large fields into several rows. In PostgreSQL this technique is called TOAST (or "the best thing since sliced bread") [36]. This means that a BLOB may not be stored continuously in a table file neither retrieved in a single go.

There were discussions [33] which approach is better for storing large binary objects: BLOBs at databases or files. The main pros for RDBMS are sophisticated indexing schemes and join algorithms. However, for large binary data types it is not justified since there are no facilities to scale them. Hence, all previous work in RDBMS research field will be lost.

Conclusions and Further Work

ChronosServer enables real-time delivery of vast amounts of data stored in various formats to thousands of concurrent clients. Using it as a backend, Climate Wikience made possible the analyzes together with interactive 3D exploration of all available climate reanalysis archives to wide audience which is not limited only to a research community alone.

The performance evaluation of ChronosServer reveals that it can withstand the expected number of clients maintaining real-time response rates.

It is straightforward to use ChronosServer to enable real-time access for numerical models output, measurements from distributed mobile and sensor networks as well as many other important retrospective data.

ChronosServer still requires some additional research concerning load balance and security. It is also beneficial to enable built-in computations during data retrieval, for example, summary statistics.

The author believes that the system will advance the international research community in understanding of climate variability and change as well as other important domains.

The Climate Wikience is freely available for download at wikience.donntu.edu.ua.

Acknowledgements

This work was supported by Award No. UKM1-2973-DO-09 of the U.S. Civilian Research & Development Foundation (CRDF). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of CRDF.

References:

1. Kalnay et al., The NCEP/NCAR 40-year reanalysis project, BAMS, 77, 437-470, 1996.

2. NCEP-DEO AMIP-II Reanalysis (R-2): M. Kanamitsu, W. Ebisuzaki, J. Woollen, S-K Yang, J.J. Hnilo, M. Fiorino, and G. L. Potter. 1631-1643, Nov 2002, Bul. of the Atmos. Met. Soc.

3. Uppala S. M. et al, The ERA-40 re-analysis, Q. J. R. Meteorol. Soc. (2005), 131, pp. 2961–3012.

4. Compo G.P. et al., The Twentieth Century Reanalysis Project, Q. J. R. Meteorol. Soc. 137:128, ,January, 2011 Part A.

5. Saha S. et al, The NCEP Climate Forecast System Reanalysis, BAMS, 1015–1057, 2011.

7. Hannachi A., Awad A., Ammar K. Climatology and classification of Spring Saharan cyclone tracks, Clim. Dyn., (37) 473–491, 2011.

8. Murray, R. J., and I. Simmonds, A numerical scheme for tracking cyclone centres from digital data. Part I: Development and operation of the scheme. Australian Meteorological Magazine, 39, 155–166, 1991.

9. Google Academy [Electronic resource] http://scholar.google.com.ua/scholar?cites=14652521284756584733&as_sdt=2005&sciodt=0,5&hl=ru

10. Introduction to Data Mining and Knowledge Discovery, Third Edition. Two Crows Corporation. [Electronic resource] http://www.twocrows.com/intro-dm.pdf

11. Rodriges Zalipynis R.A., Zapletin E.A., Averin G.V. The Wikience: Community Data Science. Concept and Implementation. Proceedings of the ICT–2011: Informatics and Computer Technologies, International Scientific-Technical Conference of Students, Postgraduate Students and Young Scientists, Donetsk, November 22–23, 2011.

12. Climate Wikience [Electronic resource] http://wikience.donntu.edu.ua

13. Apache HBase [Electronic resource] http://hbase.apache.org/

14. Rodriges Zalipynis R.A. Data and data mining methods for natural environment research. // System analysis and information technology in environmental and social sciences, Donetsk National Technical University, Vol. 1, 2011. – p. 94–107.

15. Kalnay E. Atmospheric modeling, data assimilation and predictability. – Cambridge University Press, 2003. – 369 p.

16. NCEP-DOE AMIP-II Reanalysis (AKA Reanalysis 2) [Electronic resource] http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis2.html

17. Unidata | NetCDF [Electronic resource] http://www.unidata.ucar.edu/software/netcdf/

18. GRIB.US > Home [Electronic resource] http://www.grib.us/

19. Ghemawat S. et al. The Google File System. SOSP'03, Bolton Landing, New York, USA, 2003.

20. Bajda-Pawlikowski K. et al Efcient Processing of Data Warehousing Queries in a Split Execution Environment, SIGMOD11, June12–16, 2011, Athens, Greece.

21. Chang F. et al Bigtable: A Distributed Storage System for Structured Data, OSDI 2006.

22. Cloudera Hadoop Distribution [Electronic resource] http://www.cloudera.com/hadoop/

23. Kemp, M.U., van Loon, E.E., Shamoun-Baranes, J.,and Bouten,W. RNCEP: global weather and climate data at your fingertips. Methods in Ecology and Evolution, 2011.

24. R Development Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna. [Electronic resource] http://www.r-project.org/

25. Schwan P. Lustre: Building a File System for 1,000-node Clusters. Proceedings of the Linux Symposium. – Ontario, Canada, 23–26 July 2003, p. 380–386.

26. Home | TOP500 Supercomputing Sites [Electronic resource] http://top500.org/

27. MongoDB [Electronic resource] http://www.mongodb.org/

28. GridFS Specification – MongoDB [Electronic resource] http://www.mongodb.org/display/DOCS/GridFS+Specification

29. Redis [Electronic resource] http://redis.io/

30. Membase [Electronic resource] http://www.couchbase.org/membase

31. MySQL Cluster [Electronic resource] http://www.mysql.com/products/cluster/

32. Vertica [Electronic resource] http://www.vertica.com/

33. Sears R., Ingen C., Gray J. To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem, Technical Report MSR-TR-2006-45, 2006.

34. The rasdaman raster array database [Electronic resource] http://rasdaman.eecs.jacobs-university.de/trac/rasdaman

35. Tera-Scale Image Processing | Jacobs University [Electronic resource] http://www.jacobs-university.de/lsis/projects/TeraPro

36. PostgreSQL: Documentation: Manuals: PostgreSQL 8.3: TOAST [Electronic resource] http://www.postgresql.org/docs/8.3/static/storage-toast.html

Надійшла до редколегії 08.10.2011

Р.А. РОДРИГЕС ЗАЛЕПИНОС
Донецкий национальный технический университет

ChronosServer: доступ в реальном времени тысяч одновременных клиентов к "нативному" многотерабайтному ретроспективному хранилищу данных

ChronosServer работает на компьютерном кластере, который построен на оборудовании широкого потребления и обладает свойствами масштабируемости, высокой доступности и отказоустойчивости. Он создает на основе уже существующих больших объемов данных деятельный интеллектуальный продукт не изменяя исходные файлы. ChronosServer обнаруживает файлы на узлах кластера, анализирует их структуру и предоставляет независимую от формата SQL-подобную модель запросов для доступа к их содержимому. Он способен напрямую читать сжатые данные из различных форматов, включая NetCDF, GeoTIFF, GRIB, HDF и многих других. Это полностью сохраняет метаданные, хранящиеся в файле, в оригинальном виде, что необходимо для их корректной интерпретации и обработки другим программным обеспечением. Новые данные добавляются в систему прозрачным plug-and-play образом простым копированием их на узел кластера, сокращая затраты на администрирование. Это позволяет существующему программному обеспечению, например, ГИС системам либо статистическим пакетам напрямую оперировать с файлами, которые используются ChronosServer, а также не изменять старые коды генерации данных. ChronosServer сохраняет действующую на данный момент инфраструктуру неизменной, избегая болезненные, трудоемкие и подверженные ошибкам процедуры конвертации файлов данных, предоставляя в то же время дополнительные возможности для их анализа.

Хранилище данных, доступ в реальном времени, форматы файлов, метаданные, неизменная действующая инфраструктура, старые коды

Р.А. РОДРІГЕС ЗАЛІПИНІС
Донецький національний технічний університет

ChronosServer: доступ у реальному часі тисяч одночасних клієнтів до "нативного" багатотерабайтного ретроспективного сховища даних

ChronosServer працює на комп'ютерному кластері, який побудовано на обладнані широкого спожитку та має властивості масштабованості, високої доступності та відмовостійкості. Він створює на основі вже існуючих великих об'ємів даних діяльний інтелектуальний продукт не змінюючи вхідні файли. ChronosServer виявляє файли на вузлах кластеру, аналізує їхню структуру та надає незалежну від формату SQL-подібну модель запитів для доступу до їхнього вмісту. Він здатен напряму читати стислі дані з різних форматів, включаючи NetCDF, GeoTIFF, GRIB, HDF та багатьох інших. Це повністю зберігає метадані, які знаходяться у файлі у оригінальному вигляді, що необхідно для їхньої коректної інтерпретації та обробки іншим програмним забезпеченням. Нові дані додаються у систему прозорим plug-and-play методом у вигляді простого копіювання їх на вузол кластеру, що зменшує затрати на адміністрування. Це дозволяє існуючому програмному забезпеченню, наприклад, ГІС системам або статистичним пакетам напряму оперувати з файлами, які використовуються ChronosServer, а також не модифікувати старі коди генерації даних. ChronosServer зберігає діючу на даний момент інфраструктуру незмінною, запобігаючи трудомісткі та піддані помилкам процедури конвертації файлів даних, надаючи у той самий час додаткові можливості для їхнього аналізу.

Сховище даних, доступ у реальному часі, формати файлів, метадані, незмінна діюча інфраструктура, старі коди