Article
Alexey Maslov · Oct 20, 2020 11m read

Parallel Processing of Multi-Model Data in InterSystems IRIS and Caché

As we all well know, InterSystems IRIS has an extensive range of tools for improving the scalability of application systems. In particular, much has been done to facilitate the parallel processing of data, including the use of parallelism in SQL query processing and the most attention-grabbing feature of IRIS: sharding. However, many mature developments that started back in Caché and have been carried over into IRIS actively use the multi-model features of this DBMS, which are understood as allowing the coexistence of different data models within a single database. For example, the HIS qMS database contains both semantic relational (electronic medical records) as well as traditional relational (interaction with PACS) and hierarchical data models (laboratory data and integration with other systems). Most of the listed models are implemented using SP.ARM's qWORD tool (a mini-DBMS that is based on direct access to globals). Therefore, unfortunately, it is not possible to use the new capabilities of parallel query processing for scaling, since these queries do not use IRIS SQL access.

Meanwhile, as the size of the database grows, most of the problems inherent to large relational databases become right for non-relational ones. So, this is a major reason why we are interested in parallel data processing as one of the tools that can be used for scaling.

In this article, I would like to discuss those aspects of parallel data processing that I have been dealing with over the years when solving tasks that are rarely mentioned in discussions of Big Data. I am going to be focusing on the technological transformation of databases, or, rather, technologies for transforming databases.

It's no secret that the data model, storage architecture, and software and hardware platform are usually chosen at the early stages of system development, often when the project is still far from completion. However, some time will pass, and it is quite often the case when several years after the system was deployed, the data needs to be migrated for one reason or another. Here are just a few of the commonly encountered tasks (all examples are taken from real life):

1. A company is planning to go international, and its database with 8-bit encoding must be converted to Unicode.
2. An outdated server is being replaced with a new one, but it is either impossible to seamlessly transfer journals between servers (using Mirroring or Shadowing IRIS system features) due to licensing restrictions or lack of capabilities to meet existing needs, such as, for example, when you are trying to solve task (1).
3. You find that you need to change the distribution of globals among databases, for example, moving a large global with images to a separate database.

You might be wondering what is so difficult about these scenarios. All you need to do is stop the old system, export the data, and then import it into the new system. But if you are dealing with a database that is several hundred gigabytes (or even several terabytes) in size, and your system is running in 7x24 mode, you won't be able to solve any of the mentioned tasks using the standard IRIS tools.

Basic approaches to task parallelization

"Vertical" parallelization

Suppose that you could break a task down into several component tasks. If you are lucky, you find out that you can solve some of them in parallel. For example,

- Preparing data for a report (calculations, data aggregation)
- Applying style rules
- Printing out reports

can all be performed at the same time for several reports: one report is still in the preparation phase, another is already being printed out at the same time, etc. This approach is nothing new. It has been used ever since the advent of batch data processing, that is, for the last 60 years. However, even though it is not a new concept, it is still quite useful. However, you will only realize a noticeable acceleration effect when all subtasks have a comparable execution time, and this is not always the case.

"Horizontal" parallelization

When the order of operations for solving a task consists of iterations that can be performed in an arbitrary order, they can be performed at the same time. For example:

- Contextual search in the global:
  - You can split the global into sub-globals ($order by the first index).
  - Search separately in each of them.
  - Assemble the search results.
- Transferring the global to another server via a socket or ECP:
  - Break the global into parts.
  - Pass each of them separately.

Common features of these tasks:

- Identical processing in subtasks (down to sharing the same parameters),
- The correctness of the final result does not depend on the order of execution of these subtasks,
- There is a weak connection between the subtasks and the "parent" task only at the level of result reporting, where any required postprocessing is not a resource-intensive operation.

These simple examples suggest that horizontal parallelism is natural for data transformation tasks, and indeed, it is. In what follows, we will mainly focus on this type of parallel processing.

"Horizontal" parallelization

One of the approaches: MapReduce

MapReduce is a distributed computing model that was introduced by Google. It is also used to execute such operations to process large amounts of information at the same time. Popular open source implementations are built on a combination of Apache Hadoop (https://ru.bmstu.wiki/Apache_Hadoop) and Mahouthttps://ru.bmstu.wiki/Apache_Mahout.

Basic steps of the model: Map (distribution of tasks between handlers), the actual processing, and Reduce (combining the processing results).

For the interested reader who would like to know more, I can recommend the series of articles by Timur Safin on his approach to creating the MapReduce tool in IRIS/Caché, which starts with Caché MapReduce - an Introduction to BigData and the MapReduce Concept (Part I).

Note that due to the "innate ability" of IRIS to write data to the database quickly, the Reduce step, as a rule, turns out to be trivial, as in the distributed version of WordCount. When dealing with database transformation tasks, it may turn out to be completely unnecessary. For example, if you used parallel handlers to move a large global to a separate database, then we do not need anything else.

How many servers?

The creators of parallel computing models, such as MapReduce, usually extend it to several servers, the so-called data processing nodes, but in database transformation tasks, one such node is usually sufficient. The fact of the matter is that it does not make sense to connect several processing nodes (for example, via the Enterprise Cache Protocol (ECP)), since the CPU load required for data transformation is relatively small, which cannot be said about the amount of data involved in processing. In this case, the initial data is used once, which means that you should not expect any performance gain from distributed caching.

Experience has shown that it is often convenient to use two servers whose roles are asymmetric. To simplify the picture somewhat:

- The source database is mounted on one server (*Source DB*).
- The converted database is mounted on the second server (*Destination DB*).
- Horizontal parallel data processing is configured only on one of these servers; the operating processes on this server are the *master* processes to
- the processes running on the second server, which are, in turn, the *slave* processes; when you use ECP, these are the DBMS system processes (ECPSrvR, ECPSrvW, and ECPWork), and when using a socket-oriented data transfer mechanism, these are the child processes of TCP connections.

We can say that this approach to distributing tasks combines horizontal parallelism (which is used to distribute the load within the master server) with vertical parallelism (which is used to distribute "responsibilities" between the master and slave servers).

Tasks and tools

Let us consider the most general task of transforming a database: transferring all or part of the data from the source database to the destination database while possibly performing some kind of re-encoding of globals (this can be a change of encoding, change of collation, etc.). In this case, the old and new databases are local on different database servers. Let us list the subtasks to be solved by the architect and developer:

1. Distribution of roles between servers.
2. Choice of data transmission mechanism.
3. Choice of the strategy for transferring globals.
4. Choice of the tool for distributing tasks among several processes.

Let's take a bird's eye view of them.

Distribution of roles between servers

As you are already well familiar with, even if IRIS is being installed with Unicode support, it is also able to mount 8-bit databases (both local and remote). However, the opposite is not true: the 8-bit version of IRIS will not work with a Unicode database, and there will be inevitable <WIDE CHAR> errors if you try to do so. This must be taken into account when deciding which of the servers – the source or the destination – will be the master if the character encoding is changed during data transformation. However, it is impossible here to decide on an ultimate solution without considering the next task, which is the

Choice of data transmission mechanism

You can choose from the following options here:

1. If the license and DBMS versions on both servers allow the ECP to be used, consider ECP as a transport.
2. If not, then the simplest solution is to deal with both databases (the source and destination) locally on the destination system. To do this, the source database file must be copied to the appropriate server via any

available file transport, which, of course, will take additional time (for copying the database file over the network) and space (to store a copy of the database file).
3. In order to avoid wasting time copying the file (at least), you can implement your mechanism for exchanging data between the server processes via a TCP socket. This approach may be useful if:
    - The ECP cannot be used for some reason, e.g. due to the incompatibility of the DBMS versions serving the source and destination databases (for example, the source DBMS is of a very legacy version),
    - Or: It is impossible to stop users from working on the source system, and therefore the data modification in the source database that occurs in the process of being transferred must be reflected in the destination database.

My priorities when choosing an approach are pretty evident: if the ECP is available and the source database remains static while it is transferred – 1, if the ECP is not available, but the database is still static – 2, if the source database is modified – 3. If we combine these considerations with master server choice, then we can produce the following possibility matrix:

Is the source DB static during transmission?	**Is the ECP protocol available?**	**Location of the DB source**	**Master system**
Yes	Yes	Remote on the target system	Target
Yes	No	Local (copy) on the target system	Target
No	It does not matter, as we will use our mechanism for transferring data over TCP sockets.	Local (original) on the source system	Source

Choice of the strategy for transferring globals

At first glance, it might seem that you can simply pass globals one by one by reading the Global Directory. However, the sizes of globals in the same database can vary greatly: I recently encountered a situation when globals in a production database ranged in size between 1 MB and 600 GB. Let's imagine that we have nWorkers worker processes at our disposal, and there is at least one global ^Big for which it is true:

Size(^Big) > (Summary Size of All ^Globals) / nWorkers

Then, no matter how successfully the task of transferring the remaining globals is distributed between the working processes, the task that ends up being assigned the transfer of the ^Big global will remain busy for the remaining allocated part of the time and will probably only finish its task long after the other processes finish processing the rest of the globals. You can improve the situation by pre-ordering the globals by size and starting the processing with the largest ones first, but in cases where the size of ^Big deviates significantly from the average value for all globals (which is a typical case for the MIS qMS database):

Size(^Big) >> (Summary Size of All ^Globals) / nWorkers

this strategy will not help you much, since it inevitably leads to a delay of many hours. Hence, you cannot avoid splitting large global into parts to allow its processing using several parallel processes. I wish to emphasize that this task (number 3 in my list) turned out to be the most difficult among others being discussed here, and took most of my (rather than CPU!) time to solve it.

Choice of the tool for distributing tasks among several processes

The way that we interact with the parallel processing mechanism can be described as follows:

- We create a pool of background worker processes.
- A queue is created for this pool.
- The initiator process (let's call it the *local manager*), having a plan that was prepared in advance in step 3, places *work units* in the queue; as a rule, the *work unit* comprises the name and actual arguments of a certain class method.

- The worker processes retrieve work units from the queue and perform processing, which boils down to calling a class method with the actual arguments that are passed to the work units.
- After receiving confirmation from all worker processes that the processing of all queued work units is complete, the local manager releases the worker processes and finalizes processing if required.

Fortunately, IRIS provides an excellent parallel processing engine that fits nicely into this scheme, which is implemented in the %SYSTEM.WorkMgr class. We will use it in a running example that we will explore across a planned series of articles.

In the next article, I plan to focus on clarifying the solution to task number 3 in more detail.

In the third article, which will appear if you will show some interest in my writing, I will talk about the nuances of solving task number 4, including, in particular, about the limitations of %SYSTEM.WorkMgr and the ways of overcoming them.

#Big Data #DevOps #Caché #InterSystems IRIS

Source URL: https://community.intersystems.com/post/parallel-processing-multi-model-data-intersystems-iris-and-cach%C3%A9