Python, PyTables, Java - tying all together

Asked 15 years ago Modified 2 years, 9 months ago Viewed 3k times

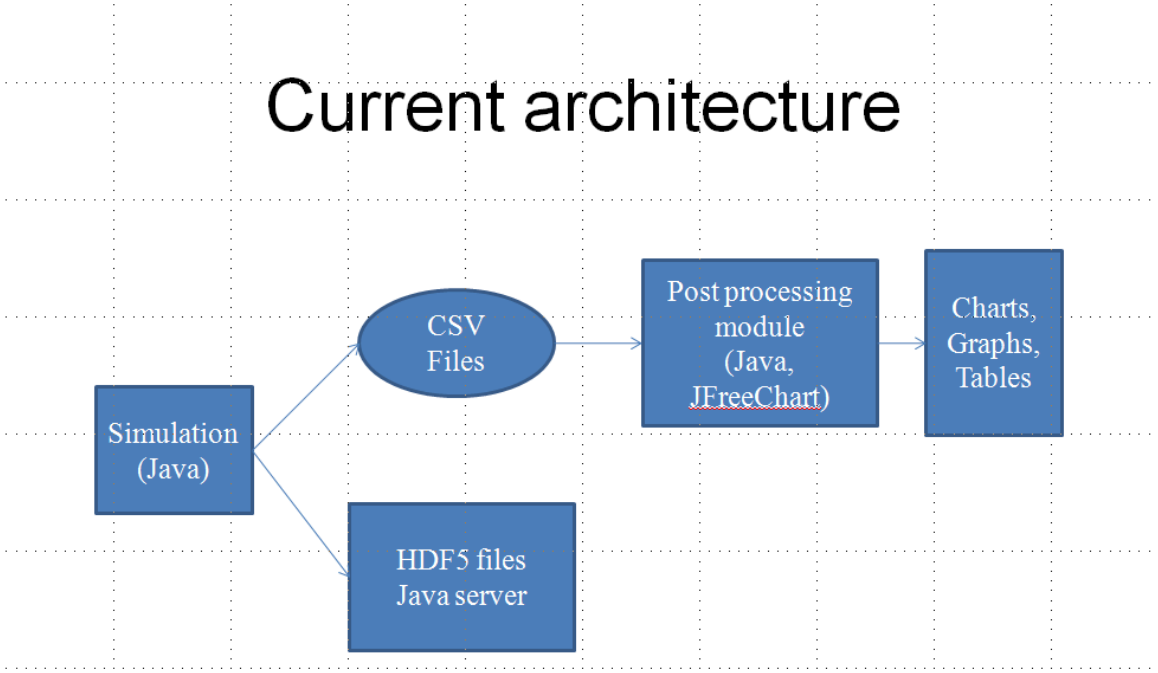
26

Question in nutshell

What is the best way to get Python and Java to play nice with each other?

More detailed explanation

I have a somewhat complicated situation. I'll try my best to explain both in pictures and words. Here's the current system architecture:



We have an agent-based modeling simulation written in Java. It has options of either writing locally to CSV files, or remotely via a connection to a Java server to an HDF5

file. Each simulation run spits out over a gigabyte of data, and we run the simulation dozens of times. We need to be able to aggregate over multiple runs of the same scenario (with different random seeds) in order to see some trends (e.g. min, max, median, mean). As you can imagine, trying to move around all these CSV files is a nightmare; there are multiple files produced per run, and like I said some of them are enormous. That's the reason we've been trying to move towards an HDF5 solution, where all the data for a study is stored in one place, rather than scattered across dozens of plain text files. Furthermore, since it is a binary file format, it should be able to get significant space savings as compared to uncompressed CSVS.

As the diagram shows, the current post-processing we do of the raw output data from simulation also takes place in Java, and reads in the CSV files produced by local output. This post-processing module uses JFreeChart to create some charts and graphs related to the simulation.

The Problem

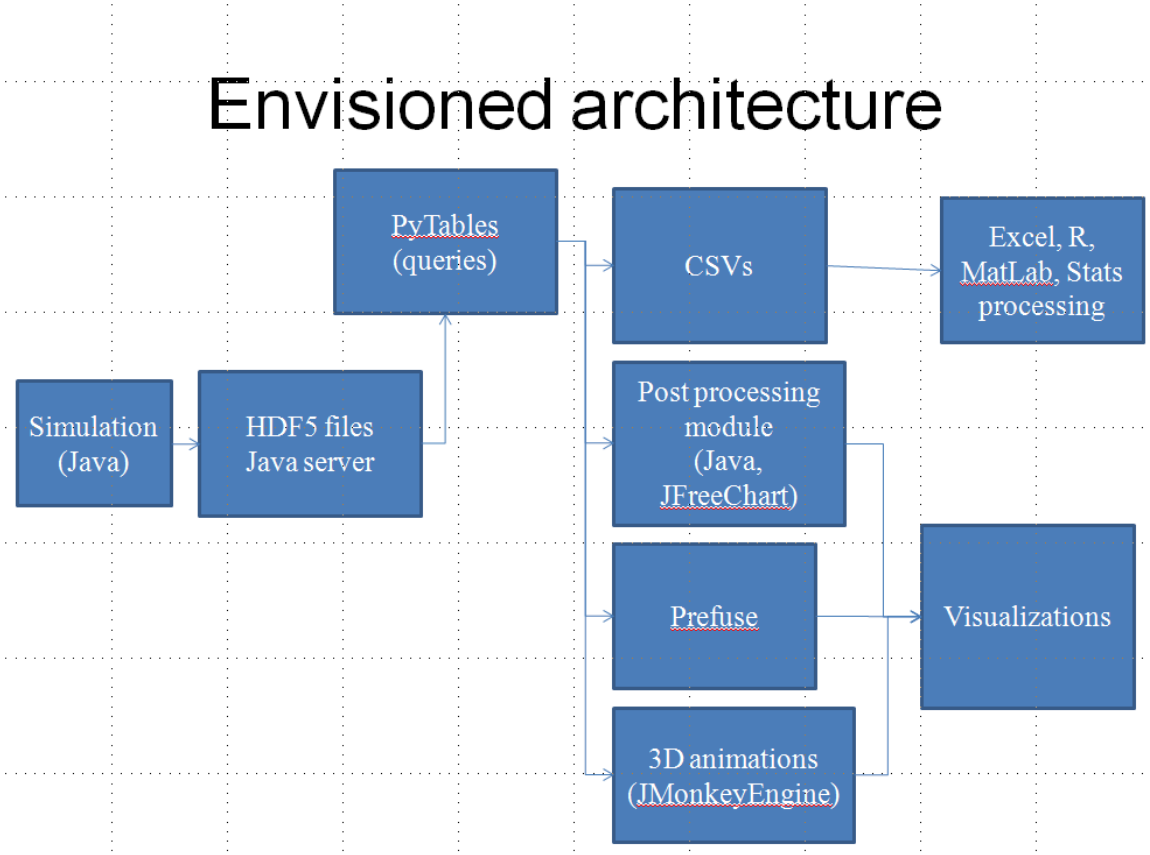
As I alluded to earlier, the CSVs are really untenable and are not scaling well as we generate more and more data from simulation. Furthermore, the post-processing code is doing more than it should have to do, essentially performing the work of a very, very poor man's relational database (making joins across 'tables' (csv files) based on foreign keys (the unique agent IDs). It is also difficult in

this system to visualize the data in other ways (e.g. Prefuse, Processing, JMonkeyEngine getting some subset of the raw data to play with in MatLab or SPSS).

Solution?

My group decided we really need a way of filtering and querying the data we have, as well as performing cross table joins. Given this is a write-once, read-many situation, we really don't need the overhead of a real relational database; instead we just need some way to put a nicer front end on the HDF5 files. I found a few papers about this, such as one describing how to use [XQuery as the query language on HDF5 files][3], but the paper describes having to write a compiler to convert from XQuery/XPath into the native HDF5 calls, way beyond our needs. Enter [PyTables][4]. It seems to do exactly what we need (provides two different ways of querying data, either through Python list comprehension or through [in-kernel (C level) searches][5].

The proposed architecture I envision is this:



What I'm not really sure how to do is to link together the python code that will be written for querying, with the Java code that serves up the HDF5 files, and the Java code that does the post processing of the data. Obviously I will want to rewrite much of the post-processing code that is implicitly doing queries and instead let the excellent PyTables do this much more elegantly.

Java/Python options

A simple google search turns up a few options for [communicating between Java and Python][7], but I am so new to the topic that I'm looking for some actual expertise and criticism of the proposed architecture. It seems like the Python process should be running on same machine as the Datahose so that the large .h5 files

do not have to be transferred over the network, but rather the much smaller, filtered views of it would be transmitted to the clients. [Pyro][8] seems to be an interesting choice - does anyone have experience with that?

java python architecture hdf5 pytables

Share

Improve this question

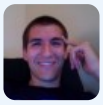
Follow

edited Mar 20, 2022 at 19:07

Glorfindel

22.6k ● 13 ● 89 ● 116

asked Dec 23, 2009 at 16:14



I82Much

27.3k ● 14 ● 89 ● 119

2 I don't know enough about Python to answer, but I want to say that this is a **beautifully written and illustrated** question. All I can offer is a suggestion that if you use Jython, you can call Java code and libraries from Python, and I believe also Jython code fro Java. Also, although HDF5 is a wonderfully designed format for large datasets, if your data is truly relational and your analytics complex, a DB would still make more sense for the ease of SQL queries. – BobMcGee Dec 23, 2009 at 16:22

1 +1 great question, very well articulated – Simon Dec 23, 2009 at 16:26

1 ...perhaps a naive question, but why do you have to have Java interact with Python? You seem to have a nicely layered architecture which provides a data interface expressed by your HDF5 file. It's not clear to me from your question what

you want to do in Java from Python - or vice versa - and why. – Simon Dec 23, 2009 at 16:30

Relational databases don't have to have high overhead. Have you looked at something like SQLite? – Ichorus Dec 23, 2009 at 16:36

Re: Ichorus: I had suggested using a RDBMS like SQLite if the HDF5 datahose plan failed, but we never got to the point of implementing it to see its performance. It just seems like overkill when we don't need any of the row-level locking or transactional features a full RDBMS – I82Much Dec 23, 2009 at 16:49

4 Answers

Sorted by: Highest score (default)

13

This is an epic question, and there are lots of considerations. Since you didn't mention any specific performance or architectural constraints, I'll try and offer the best well-rounded suggestions.

The initial plan of using PyTables as an intermediary layer between your other elements and the datafiles seems solid. However, one design constraint that wasn't mentioned is one of the most critical of all data processing: Which of these data processing tasks can be done in batch processing style and which data processing tasks are more of a live stream.

This differentiation between "we know exactly our input and output and can just do the processing" (batch) and "we know our input and what needs to be available for something else to ask" (live) makes all the difference to

an architectural question. Looking at your diagram, there are several relationships that imply the different processing styles.

Additionally, on your diagram you have components of different types all using the same symbols. It makes it a little bit difficult to analyze the expected performance and efficiency.

Another contraint that's significant is your IT infrastructure. Do you have high speed network available storage? If you do, intermediary files become a brilliant, simple, and fast way of sharing data between the elements of your infrastructure for all batch processing needs. You mentioned running your PyTables-using-application on the same server that's running the Java simulation. However, that means that server will experience load for both writing and reading the data. (That is to say, the simulation environment could be affected by the needs of unrelated software when they query the data.)

To answer your questions directly:

- PyTables looks like a nice match.
- There are many ways for Python and Java to communicate, but consider a language agnostic communication method so these components can be changed later if necessarily. This is just as simple as finding libraries that support both Java and Python and trying them. The API you choose to implement

with whatever library should be the same anyway. (XML-RPC would be fine for prototyping, as it's in the standard library, Google's Protocol Buffers or Facebook's Thrift make good production choices. But don't underestimate how great and simple just "writing things to intermediary files" can be if data is predictable and batchable.

To help with the design process more and flesh out your needs:

It's easy to look at a small piece of the puzzle, make some reasonable assumptions, and jump into solution evaluation. But it's even better to look at the problem holistically with a clear understanding of your constraints. May I suggest this process:

- Create two diagrams of your current architecture, physical and logical.
    - On the physical diagram, create boxes for each physical server and diagram the physical connections between each.
        - Be certain to label the resources available to each server and the type and resources available to each connection.
        - Include physical hardware that isn't involved in your current setup if it might be useful. (If you have a SAN available, but aren't using it, include it in case the solution might want to.)

- On the logical diagram, create boxes for every *application* that is running in your current architecture.
    - Include relevant libraries as boxes *inside* the application boxes. (This is important, because your future solution diagram currently has PyTables as a box, but it's just a library and can't do anything on it's own.)
    - Draw on disk resources (like the HDF5 and CSV files) as cylinders.
    - Connect the applications with arrows to other applications and resources as necessary. Always draw the arrow *from* the "actor" *to* the "target". So if an app writes and HDF5 file, they arrow goes from the app to the file. If an app reads a CSV file, the arrow goes from the app to the file.
    - Every arrow must be labeled with the communication mechanism. Unlabeled arrows show a relationship, but they don't show *what* relationship and so they won't help you make decisions or communicate constraints.

Once you've got these diagrams done, make a few copies of them, and then right on top of them start to do data-flow doodles. With a copy of the diagram for each "end point" application that needs your original data, start at the simulation and end at the end point with a pretty

much solid flowing arrow. Any time your data arrow flows across a communication/protocol arrow, make notes of how the data changes (if any).

At this point, if you and your team all agree on what's on paper, then you've explained your current architecture in a manner that should be easily communicable to anyone. (Not just helpers here on stackoverflow, but also to bosses and project managers and other purse holders.)

To start planning your solution, look at your dataflow diagrams and work your way backwards from endpoint to startpoint and create a nested list that contains every app and intermediary format on the way back to the start. Then, list requirements for every application. Be sure to feature:

- What data formats or methods can this application use to communicate.
- What data does it actually want. (Is this always the same or does it change on a whim depending on other requirements?)
- How often does it need it.
- Approximately how much resources does the application need.
- What does the application do now that it doesn't do that well.
- What can this application do now that would help, but it isn't doing.

If you do a good job with this list, you can see how this will help define what protocols and solutions you choose. You look at the situations where the data crosses a communication line, and you compare the requirements list for *both sides* of the communication.

You've already described one particular situation where you have quite a bit of java post-processing code that is doing "joins" on tables of data in CSV files, thats a "do now but doesn't do that well". So you look at the other side of that communication to see if the other side can do that thing well. At this point, the other side is the CSV file and before that, the simulation, so no, there's nothing that can do that better in the current architecture.

So you've proposed a new Python application that uses the PyTables library to make that process better. Sounds good so far! But in your next diagram, you added a bunch of other things that talk to "PyTables". Now we've extended past the understanding of the group here at StackOverflow, because we don't know the requirements of those other applications. But if you make the requirements list like mentioned above, you'll know exactly what to consider. Maybe your Python application using PyTables to provide querying on the HDF5 files can support all of these applications. Maybe it will only support one or two of them. Maybe it will provide live querying to the post-processor, but periodically write intermediary files for the other applications. We can't tell, but with planning, you can.

Some final guidelines:

- **Keep things simple!** The enemy here is complexity. The more complex your solution, the more difficult the solution to implement and the more likely it is to fail. Use the least number operations, use the least complex operations. Sometimes just one application to handle the queries for all the other parts of your architecture is the simplest. Sometimes an application to handle "live" queries and a separate application to handle "batch requests" is better.
- **Keep things simple!** It's a big deal! Don't write anything that can already be done for you. (This is why intermediary files can be so great, the OS handles all the difficult parts.) Also, you mention that a relational database is too much overhead, but consider that a relational database also comes with a very expressive and well-known query language, the network communication protocol that goes with it, *and* you don't have to develop anything to use it! Whatever solution you come up with has to be *better* than using the off-the-shelf solution that's going to work, for certain, very well, or it's not the best solution.
- **Refer to your physical layer documentation frequently** so you understand the resource use of your considerations. A slow network link or putting too much on one server can both rule out otherwise good solutions.

- **Save those docs.** Whatever you decide, the documentation you generated in the process is valuable. Wiki-them or file them away so you can whip them out again when the topic come s up.

And the answer to the direct question, "How to get Python and Java to play nice together?" is simply "use a language agnostic communication method." The truth of the matter is that Python and Java are both not important to your describe problem-set. What's important is the data that's flowing through it. Anything that can easily and effectively share data is going to be just fine.

Share Improve this answer

Follow

answered Dec 23, 2009 at 17:27

Travis Bradshaw

4,310 ● 3 ● 23 ● 20

Agreed. Without knowing more about what the PyTables layer is required to do, it's hard to know what the questioner needs. Some use cases would be helpful. – AFoglia Dec 23, 2009 at 18:10

+1 for the most complete, comprehensive response I've ever seen. You are right of course, I should be thinking of a way to decouple the systems from each other and program to a program agnostic interface, so that things can be more easily modified down the road. I apologize for the wonky chart; I'm new to this and have not made too many charts/graphs before to illustrate components of a complex system. I will definitely take your advice, modify the architecture diagrams, and post my results here, so that others can benefit as well. – I82Much Dec 23, 2009 at 18:26

Damn, I consider this answer relevant, thanks @Travis
– Darren Jun 5 at 9:16

5

Do not make this more complex than it needs to be.

Your Java process can -- simply -- spawn a separate subprocess to run your PyTables queries. Let the Operating System do what OS's do best.

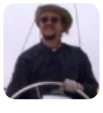
Your Java application can simply fork a process which has the necessary parameters as command-line options. Then your Java can move on to the next thing while Python runs in the background.

This has HUGE advantages in terms of concurrent performance. Your Python "backend" runs concurrently with your Java simulation "front end".

Share Improve this answer Follow

edited Dec 23, 2009 at 17:09

answered Dec 23, 2009 at 16:32



S.Lott
391k ● 82 ● 517 ● 788

0

You could try Jython, a Python interpreter for the JVM which can `import` Java classes.

Jython project homepage

Unfortunately, that's all I know on the subject.

Share Improve this answer

Follow

answered Dec 23, 2009 at 16:19

badp

11.8k ● 5 ● 63 ● 89

> PyTables requires NumPy, which AFAIK won't work with Jython. He could probably access his Java HDF5 library from Jython though. – AFoglia Dec 23, 2009 at 18:04

0

Not sure if this is good etiquette. I couldn't fit all my comments into a normal comment, and the post has no activity for 8 months.

Just wanted to see how this was going for you? We have a very very very similar situation where I work - only the simulation is written in C and the storage format is binary files. Every time a boss wants a different summary we have to make/modify handwritten code to do summaries. Our binary files are about 10 GB in size and there is one of these for every year of the simulation, so as you can imagine, things get hairy when we want to run it with different seeds and such.

I've just discovered pyTables and had a similar idea to yours. I was hoping to change our storage format to hdf5 and then run our summary reports/queries using pytables. Part of this involves joining tables from each year. Have you had much luck doing these types of "joins" using pytables?

Share Improve this answer

Follow

answered Aug 21, 2010 at 22:55

oob

1,958 4 26 48

1 Actually I moved onto a different project before having a chance to implement any of the ideas in this post. C'est la vie. I don't think the other people on the team went forward with the idea of using PyTables. I still think it's worthwhile. – I82Much Aug 22, 2010 at 2:10