Can Hadoop + GPU Boost the Performance of Your Big Data Project?

Hadoop, an open source framework that enables distributed computing, has changed the way we deal with big data. Parallel processing with this set of tools can improve performance several times over. The question is, can we make it work even faster?

By Vladimir Starostenkov

Network World |

24 JUNE 2013 23:00 SGT

Hadoop, an open source framework that enables distributed computing, has changed the way we deal with big data. Parallel processing with this set of tools can improve performance several times over. The question is, can we make it work even faster? What about offloading calculations from a CPU to a graphics processing unit (GPU) designed to perform complex 3D and mathematical tasks? In theory, if the process is optimized for parallel computing, a GPU could perform calculations 50-100 times faster than a CPU.

This article, written by the R&D team of Altoros Systems, a big data specialist and platform-as-a-service enabler, explores what is possible and how you can try this for your large-scale system.

[2.0 RELEASE: Get ready for a flood of new Hadoop apps]

The idea itself isn't new. For years scientific projects have tried to combine the Hadoop or MapReduce approach with the capacities of a GPU. Mars seems to be the first successful MapReduce framework for graphics processors. The project achieved a 1.5x-16x increase in performance when analyzing Web data (search/logs) and processing Web documents (including matrix multiplication).

Following Mars' groundwork, other scientific institutions developed similar tools to accelerate their data-intensive systems. Use cases included molecular dynamics, mathematical modeling (e.g., the Monte Carlo method), block-based matrix multiplication, financial analysis, image processing, etc.

On top of that, there is BOINC, a fast-evolving, volunteer-driven middleware system for grid computing. Although it does not use Hadoop, BOINC has already become a foundation for accelerating many scientific projects. For instance, GPUGRID relies on BOINC's GPU and distributed computing to perform molecular simulations that help "to understand the function of proteins in health and disease." Most of other BOINC projects related to medicine, physics, mathematics, biology, etc. could be implemented using Hadoop+GPU, too.

So, the demand for accelerating parallel computing systems with GPUs does exist. Institutions invest in supercomputers with GPUs or develop their own solutions. Hardware vendors, such as Cray, have released machines equipped with GPUs and pre-configured with Hadoop. Amazon has also launched Elastic MapReduce (Amazon EMR), which enables Hadoop on its cloud servers with GPUs.

But does one size fit all? Supercomputers provide the highest possible performance, yet cost millions of dollars. Using Amazon EMR is feasible only in projects that last for several months. For larger scientific projects (two to three years), investing in your own hardware may be more cost-effective. Even if you increase the speed of calculations using GPU within your Hadoop cluster, what about performance bottlenecks related to data transfer? Let's explore this in detail.

How it works

Data processing implies data exchange between HDD, DRAM, CPU and GPU. Figure 1 shows how data is transferred when a commodity machine performs computations with a CPU and a GPU.

Data exchange between components of a commodity computer when executing a task

Arrow A: Transferring data from an HDD to DRAM (a common initial step for both CPU and GPU computing)

Arrow B: Processing data with a CPU (transferring data: DRAM ' chipset ' CPU)

Arrow C: Processing data with a GPU (transferring data: DRAM ' chipset ' CPU ' chipset ' GPU ' GDRAM ' GPU)

As a result, the total amount of time we need to complete any task includes:

the amount of time required for a CPU or a GPU to carry out computations

plus the amount of time spent on data transfer between all of the components

According to Tom's Hardware (CPU Charts 2012), performance of an average CPU ranges from 15 to 130 GFLOPS. At the same time, performance of Nvidia GPUs, for instance, varies within a range of 100-3,000+ GFLOPS (2012 comparison). All of these measurements are approximate and largely depend on the type of task and the algorithm. Anyway, in some scenarios, a GPU can speed up computations by nearly five to 25 times per node. Some developers claim that if your cluster consists of several nodes, performance can be accelerated by 50x-200x. For example, the creators of the MITHRA project achieved a 254x increase.

However, what about the impact of data transfer? Different types of hardware transfer data at different speeds. Although supercomputers are most likely optimized for working with GPUs, a regular computer or server may be much slower when exchanging data.

While the rate of transferring data between an average CPU and a chipset is 10-20GBps (see Point Y on Figure 1), a GPU exchanges data with DRAM at the speed of 1-10GBps (see Point X). Although some systems may reach up to 10 GBps (PCIe v3), in most standard configurations data flows between a GPU's DRAM (GDRAM) and the DRAM of the computer at the rate of 1GBps. (It is recommended to measure the actual values on real hardware, since CPU memory bandwidth [X and Y] and the corresponding data transfer rates [C and B] can be about the same or differ by a factor of 10.)

Therefore, though a GPU provides faster computing, the main bottleneck is slow data transfer between GPU memory and CPU memory (Point X). Thus, for every particular project, you need to measure the time spent on data transfer from/to a GPU (Arrow !) against the time saved due to GPU acceleration. Therefore, the best thing is to evaluate the actual performance on a small cluster and then estimate how the system will behave on a larger scale.

You can have a look at the 2010 study by Intel that provides performance results for 14 types of exemplary use cases. According to Intel's figures, one can hardly achieve a 10x-1,000x increase in performance per single node -- instead, 2.5x or so will be more realistic. The total improvement for a cluster may be even smaller.

So, since the speed of data transfer may be rather slow, the ideal use case is when the amount of input/output data for each GPU is relatively small compared to the number of computations to be performed. It is important to keep in mind that, first, the type of the task should match the GPU's capabilities; and second, the task can be divided into parallel independent sub-processes with Hadoop.

Some examples of such tasks could include calculating complicated mathematical formulas (e.g., matrix multiplication), generating large sets of random values, similar scientific modeling tasks or other general-purpose GPU applications.

Tools to use

To create a prototype and accelerate your big data system using Hadoop coupled with GPUs, you have to use some libraries or bindings that allow for accessing a GPU. Today, the main tools you can use to employ the GPU's capabilities are as follows:

* JCUDA. The JCUDA project provides Java bindings for Nvidia CUDA and related libraries, such as JCublas, JCusparse (a library for working with matrix), JCufft (Java bindings for general signal processing), JCurand (a library for generating random numbers in GPU), etc. However, this will only work for GPUs by Nvidia.

* Java Aparapi. Aparapi converts Java bytecode to OpenCL at runtime and executes it on a GPU. Among all of the systems that use GPUs for computations with Hadoop, Aparapi and the OpenCL method seem to have the best long-term perspectives. Aparapi was developed by AMD JavaLabs, a laboratory of AMD. Released as an open-source product in 2011, the project is growing rapidly. You can take a look at some real-life use cases for this technology at the official website of the AMD Fusion Developer Summit conference.

OpenCL is an open, cross-platform standard supported by a large number of hardware vendors that allows for writing the same code base for both the CPU and the GPU. If no GPU is installed on a particular machine, OpenCL employs its CPU.

The standard is being developed by the Khronos Group, an industry consortium that includes around 100 companies such as AMD, Intel, Nvidia, Altera, Xilinx, etc. Code written with this framework can be executed on CPUs of the supported brands (AMD and Intel), as well as on GPUs manufactured by AMD and Nvidia. New solutions compatible with OpenCL appear every year, which is a big advantage.

* Creating native code to access the GPU. It is a good idea to use native code for complex mathematical computations that require a powerful GPU. The resulting performance will be much faster than in solutions that use bindings and connectors. However, if you need to deliver a solution in the shortest time possible, you may opt for frameworks like Aparapi. Then, if you are not satisfied with its performance, the original Aparapi code can be partially or completely replaced with native code. The resulting product will be considerably faster but also much less flexible.

You can use the C-language API (with Nvidia CUDA or OpenCL) to create native code that enables Hadoop to use the GPU via JNA (if your application is written in Java) or Hadoop Streaming (if your application is written in C).

GPU-Hadoop frameworks

You can also try investigating into custom GPU-Hadoop frameworks that were created after the Mars project had been launched. These include Grex, Panda, C-MR, GPMR, Shredder, SteamMR and others. However, most of them are no longer supported and were built for particular scientific projects. That means you can hardly apply a Monte Carlo simulation framework for, say, a bioinformatics project based on other algorithms.

Processor technologies are evolving, as well. You can see revolutionary new architectures in Sony PlayStation 4, Adapteva's multicore microprocessor, Mali GPU by ARM, etc. Both Adapteva and Mali GPU will be compatible with OpenCL.

Intel has also launched the Xeon Phi co-processor that works with OpenCL, too. It is a 60-core co-processor with an x86-like architecture that supports the PCI Express standard. Its performance is 1 TFLOPS in double precision with an energy consumption of just 300 Watt. This co-processor is already implemented in Tianhe-2, the most powerful supercomputer so far.

Still, it is hard to tell which architecture will become mainstream in high performance and distributed computing. In case they evolve -- and some of them certainly will -- it may change our understanding of how huge arrays of data should be processed.

Vladimir Starostenkov is a senior R&D engineer at Altoros Systems, a company that focuses on accelerating big data projects and platform-as-a-service enablement. He has more than five years of experience in implementing complex software architectures, including data-intensive systems and Hadoop-driven applications. Having strong background in computer science, Vladimir is interested in artificial intelligence and machine learning algorithms.

This story, "Can Hadoop + GPU Boost the Performance of Your Big Data Project?" was originally published by Network World.

Next read this:

- *10 most misused buzzwords in IT*
- *9 IT resolutions for 2021*
- *Top 8 challenges IT leaders will face in 2021*
- *Top 5 strategic priorities for IT leaders in 2021*

- *CIO resumes: 6 best practices and 7 strong examples*

- *11 old-school IT principles that still rule*
- *10 future trends and how CIOs can keep ahead in 2021*
- *10 lies CIOs tell themselves*
- *7 signs it's time to kill an IT investment*
- *13 'best practices' IT should avoid at all costs*

Discover what your peers are reading. Sign up for our FREE email newsletters today!

8 goals every project manager should aspire to achieve

Top 7 ERP trends for 2021

IT Resume Makeover: Cut down details to stand out as a leader

6 ways to recession-proof your IT career

SAP rolls out RPA, low-code tools to help cut dev debt