EFFECTIVE END-TO-END MANAGEMENT OF DATA ACQUISITION AND ANALYSIS FOR X-RAY PHOTON CORRELATION SPECTROSCOPY*

F. Khan †, J.P. Hammonds, S. Narayanan, A. Sandy, N. Schwarz, ANL, Argonne, IL 60439, USA

Abstract

Low latency between data acquisition and analysis is of critical importance to any experiment. The combination of a faster parallel algorithm and a data pipeline for connecting disparate components (detectors, clusters, file formats) enabled us to greatly enhance the operational efficiency of the x-ray photon correlation spectroscopy experiment facility at the Advanced Photon Source. The improved workflow starts with raw data (120 MB/s) streaming directly from the detector camera, through an on-the-fly discriminator implemented in firmware to Hadoop's distributed file system in a structured HDF5 data format. The user then triggers the MapReduce-based parallel analysis. For effective bookkeeping and data management, the provenance information and reduced results are added to the original HDF5 file. Finally, the data pipeline triggers user-specific software for visualizing the data. The whole process is completed shortly after data acquisition — a significant improvement of operation over the previous setup. The faster turnaround time helps scientists to make near real-time adjustments to the experiments.

INTRODUCTION

X-ray photon correlation spectroscopy (XPCS) is a powerful technique for characterizing the dynamic nature of complex materials over a range of time scales. The recent development of higher-frequency detectors allows the investigation of faster dynamic processes. A consequence of these detector advancements is the creation of large amounts of image data that must be processed within the time it takes to collect the next data set. Parallel computational techniques and high-performance computing (HPC) resources are required to handle this increase in data.

The dynamics in the sample is quantified using the normalized intensity autocorrelation function

$$g_2(\tau, q) = \frac{\langle I(t,q)I(t+\tau,q)\rangle}{\langle I(t,q)\rangle^2} = 1 + \beta exp\left(\frac{-\tau}{\tau_0(q)}\right),$$

where $I(t,q)$ is the intensity scattered at the momentum transfer q (inverse length scale) at time t, and τ_0 is the characteristic relaxation time at q. The analysis of the g_2 function allows one to determine the q-dependence of the characteristic time scale for the dynamics within the probed sample.

XPCS has been successfully applied to study a wide range of systems ranging from colloidal suspensions [1], gels, and polymers to more recent biological systems like the aging of proteins in eye-lens suspensions that are responsible for cataract formation. The workflow in use at the APS is shown in Figure 1.

ACQUISITION

The CCD-based data acquisition system [2] comprises a direct detection CCD detector operating continuously at 60 frames/s generating 120 MB/s of streaming raw data and a field-programmable gate array (FPGA) hosted on a commercial frame grabber that compresses the data into a sparse format in real time. Since the detector is operating in the photon detection mode and owing to the sparsity of the scattered signal, the purpose of data compression upstream in the workflow is to reduce the data bandwidth so the storage and real-time computation can be performed in an efficient manner. The compression algorithm is based on a good estimation of the average dark signal and noise from the detector and is performed by computing a running average and standard deviation estimation in the firmware using the FPGA. The output of the FPGA-based compression system is a series of compressed data frames each consisting of the location and the intensity of pixels that register a photon hit. This results in an effective 10-20-fold compression based on the scattering signal strength of the specimen that is being measured.

The entire data acquisition system is integrated with EPICS Area Detector, which provides a suite of plugins such as different file formats and statistical analysis. A file-saving plugin converts the compressed binary stream into the selected file format and writes to the Hadoop Distributed File System (HDFS).

MULTI-TAU AUTOCORRELATION

A new parallel implementation of the multi-tau algorithm for multi-speckle XPCS data uses the Hadoop [3] MapReduce [4] framework. This system performs analysis in two separate MapReduce phases. The map phases decompose the problem into independently solvable tasks. The reduce phases perform the computation on data from the map phases on different processor cores. Both MapReduce phases are shown in Figure 2.

The first MapReduce phase computes the multi-tau g_2 correlation. Each mapper partitions the intensity data

* Work supported by U.S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357.

† fkhan@aps.anl.gov

ISBN 978-3-95450-139-7

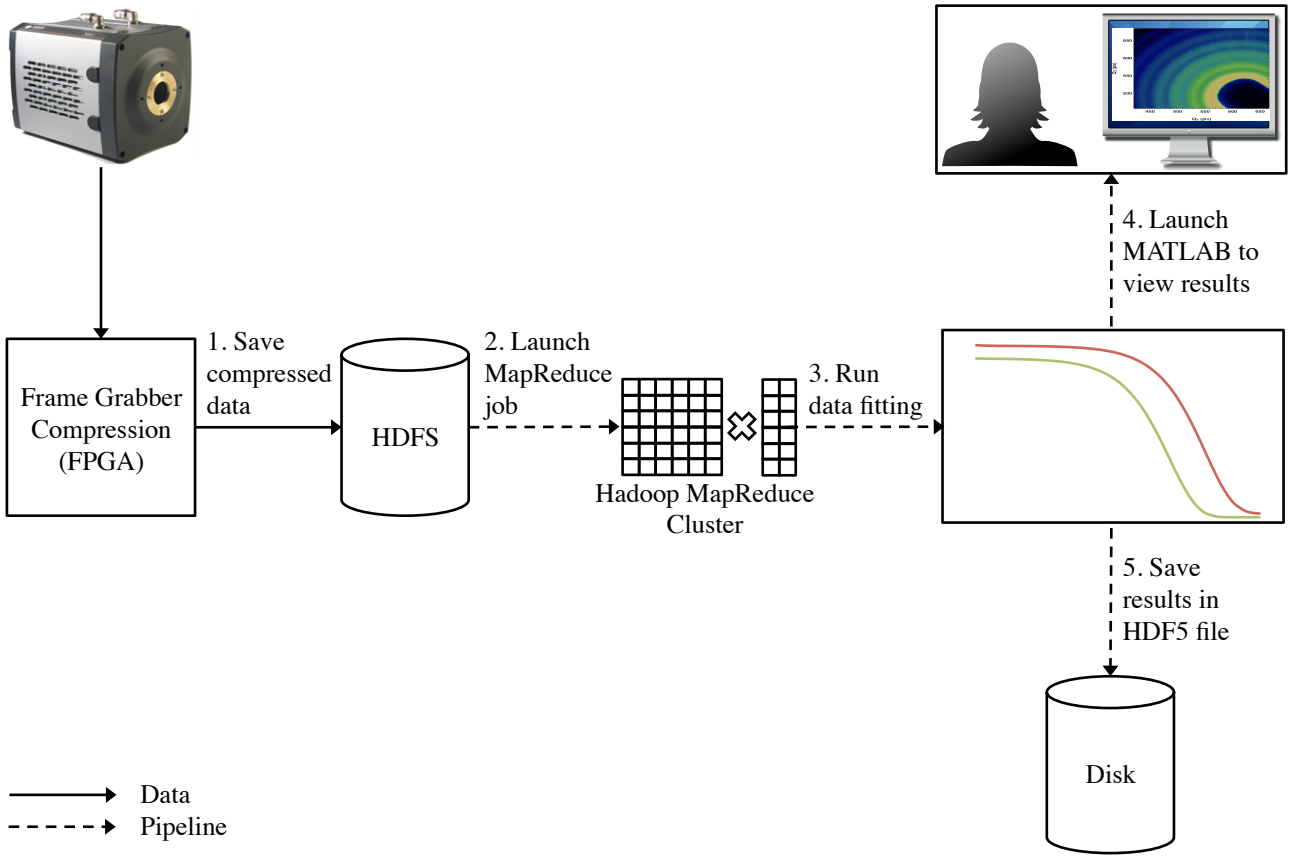

Figure 1: Different components of the XPCS acquisition and analysis workflow system in use at the APS. These components are loosely connected to each other using a well-defined HDF5 file interface. (1) The acquisition system writes data directly to the Hadoop Distributed File System (HDFS). (2) After acquisition, a message in the workflow pipeline is queued. When the message is processed, the MapReduce job that computes multi-tau autocorrelation of raw input data from the detector is launched. (3) The user has the option of running different data-fitting algorithms on the MapReduce job's output results. These are usually implemented as Python scripts. (4) The user views the output using MATLAB. (5) Results are saved in an HDF5 file on disk. The user can adjust input parameters and submit the job to be processed again.

for a single pixel location across all frames of the input dataset read from the HDFS. The sort phase moves the mapped intensity data to the appropriate processor for each reducer. The g_2 reducers receive all data with identical pixel coordinates and calculate the correlation function. The results are written back to the HDFS.

The second MapReduce phase normalizes the output from the first phase based on a user-defined pixel-binning scheme. The mappers partition the g_2 data according to the user-defined bins. The sort phase moves the mapped data for each user-defined bin to the appropriate processor for each reducer. The reducers perform the normalization for their respective bins.

HDF5

The Hierarchical Data Format 5 (HDF5) [5] is a widely used scientific data format for storing binary data along with the associated metadata in hierarchical and self-describing tags. HDF5 is the primary data format used by our system for storing both data and results. The exact structure of our HDF5 file is well defined and extensively documented in the Scientific Data Exchange format [6].

The acquisition system writes the acquisition parameters for each experiment in a single HDF5 file grouped under a top-level HDF5 dataset named "/measurement." Child elements of this dataset contain information about the experiment, such as detector type, beam energy, exposure time, etc. Any additional input from the user required for analyzing the data is written to the "/xpcs" dataset in the same HDF5 file. The analysis pipeline refers to this information when running calculations to produce the results. The final results along with the output from any intermediate analysis steps are saved under the same HDF5 dataset called "/exchange." The semantic of "/xpcs_N" and "/exchange_N" (where N is replaced with the number of the most recent input/analysis pairs) is used to keep track of multiple types of analysis that ran on the same input data. Figure 3 shows a typical data layout for an HDF5 file.

ISBN 978-3-95450-139-7

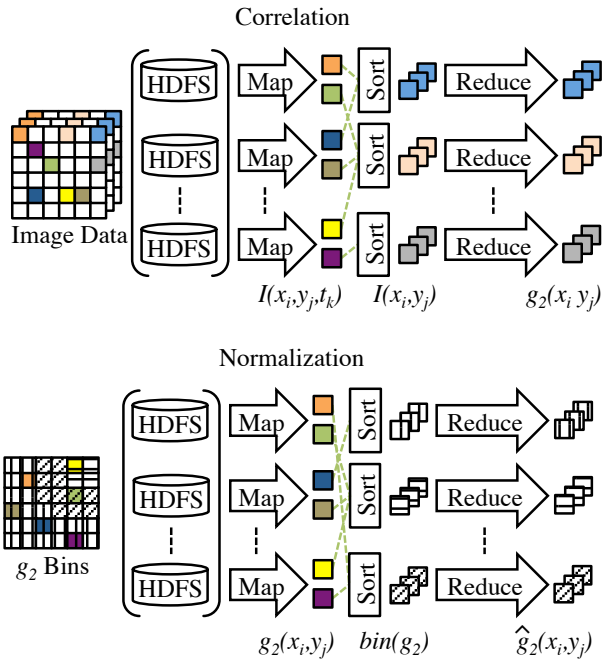

Figure 2: XPCS MapReduce phases illustrating data flow for the multi-tau autocorrelation (top) and normalization (bottom) algorithms.

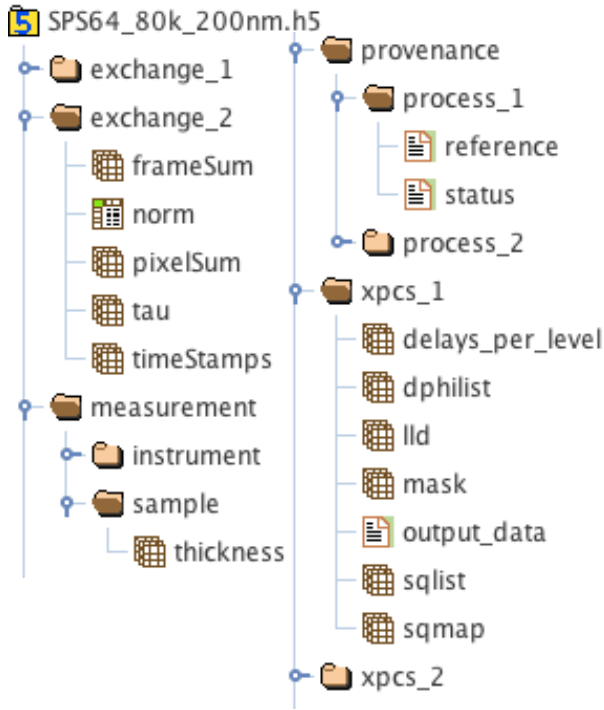

Figure 3: A sample HDF5 file showing the structure of data and result tags.

PIPELINE

Phases of this process are combined together using a workflow pipeline. It uses an industry-standard messaging system for reliable task sequencing and triggering. Generic actors handle common tasks such as file transfers. Technique-specific analysis code is implemented or called from custom actors that may be written in Java, C++, or Python. Experiment metadata and provenance information is stored along with raw and analyzed data in a single HDF5 file that is manipulated by different stages of the pipeline.

The pipeline is a series of stages connected by message queues. Stages do something; they may be the acquisition software, they may be data analysis software, or they may transfer files. Stages may be written in a variety of different programming languages, including C++, Java, and Python. Message queues are how stages are connected. When a stage is done with its task, it queues up a message for the next stage in line. That stage will eventually get to that message and process it. The message queues themselves are JMS message queues. The ActiveMQ implementation [7] of the JMS standard is used.

A stage in the pipeline is composed of two classes: the *Director* and the *Actor*. The *Director* is the interface to the message queue, in this case to the ActiveMQ implementation of JMS. It handles the core infrastructure. It only needs to be written once per programming language and then reused. The *Actor* is the part that actually does the work. It is the part that interfaces with the acquisition, runs analysis, or starts a file transfer. It may save results, and it should report status back to the pipeline so that the rest of the workflow and the user know its status.

The *Director* handles everything to do with the message queue system. This includes processing incoming and outgoing job messages, constructing signal messages for the next stage, and handling control signals for killing or restarting jobs. Additionally, the *Director* keeps a history of all messages being processed by the queuing system.

The *Actor* is a simple class definition that requires the implementation of two methods: execute and abort. Both methods should return a status to the pipeline. The status is used to determine whether to launch the next stage and displays status to the user via the pipeline's GUI. Figure 4 summarizes these components.

RESULTS

The use of an automated workflow pipeline greatly reduces the turnaround time between the acquisition system and the final computed correlation functions. This allows users to adjust experimental conditions in near realtime, thus achieving better utilization of beam time. Here we discuss the performance of individual components that make up the pipeline.

Acquisition

The system is in production use at the APS 8-ID-I beamline using a 60-Hz (soon to be 200-Hz), 1-megapixel area detector collecting up to 120,000 frames.

ISBN 978-3-95450-139-7

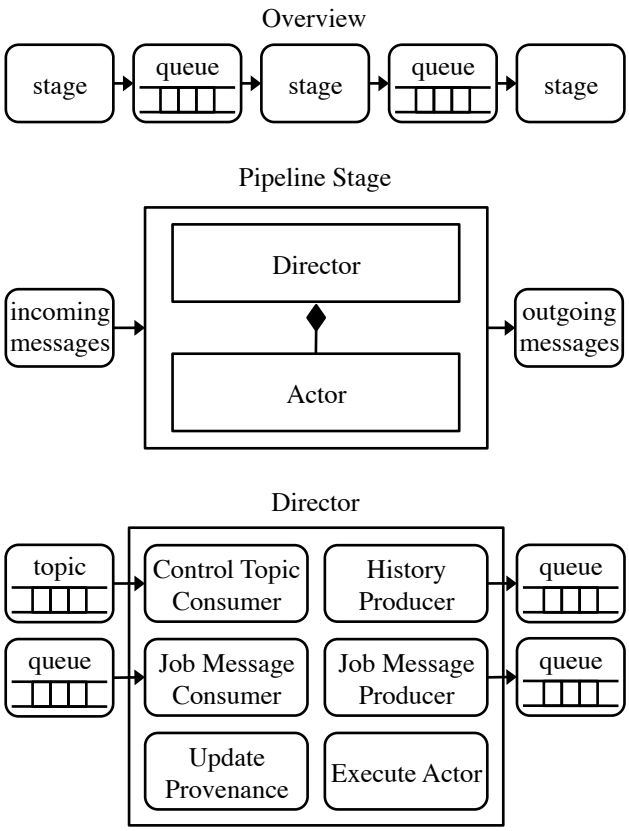

Figure 4: Overview of the pipeline architecture (top). A pipeline stage that performs an action (center). The pipeline *Director* that regulates messages and activities of the pipeline (bottom).

MapReduce

The Hadoop MapReduce implementation runs on a 96-core distributed-memory cluster with 2 GB of memory per core and 5 TB of HDFS storage space. Table 1 shows the performance of this system for datasets of varying sizes and numbers of frames. The MapReduce system scales very well as the number of frames and file sizes increase. The Hadoop MapReduce implementation replaces our previous MPI-based implementation [8, 9] for production use.

Table 1: Autocorrelation Performance Results

Size	Number of Frames	Time
45 GB	120,000	10 min
20 GB	60,000	5 min
11 GB	20,000	9 min
3.9 GB	20,000	6.5 min
0.8 GB	80,000	3 min
0.8 GB	20,000	2.5 min
0.4 GB	40,000	1 min

Workflow

The message-passing interface between different components of the workflow pipeline is based on a proven system for passing messages between distributed components at very high rates. In order to ensure maximum availability of this critical component we use the failover capabilities of ActiveMQ. A redundant server runs alongside the primary server and can take over in case the primary server goes down.

The combination of parallel analysis algorithms and a state-of-the-art messaging backend to tie together different pipeline components allow us to achieve near real-time analysis and maximize the overall efficiency of the equipment. However, there are still potentials for improvement in performance, especially related to continually increasing detector speeds.

ACKNOWLEDGMENT

We'd like to thank Mitchell McCuiston and Marcin Sikorski for their programming effort and domain knowledge, and Roger Sersted, Brian Robinson and Kenneth Sidorowicz for their technical support.

REFERENCES

[1] M. Sikorski, A.R. Sandy, and S. Narayanan, "Depletion-Induced Structure and Dynamics in Bimodal Colloidal Suspensions," Phys. Rev. Lett. 106. 188301-1 (2011).

[2] T. Madden, P. Fernandez, P. Jemian, S. Narayanan, A. R. Sandy, M. Sikorski, M. Sprung, and J. Weizeorick, "Firmware Lower-Level Discrimination and Compression Applied to Streaming X-ray Photon Correlation Spectroscopy Area-Detector Data," Rev. Sci. Instrum. 82. 075109 (2011).

[3] Hadoop, `wiki.apache.org/hadoop`.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM 51(1), (2008) 107.

[5] Hierarchical Data Format version 5 (HDF5), 2000-2010. `http://www.hdfgroup.org/HDF5`.

[6] The Scientific Data Exchange, `http://www.aps.anl.gov/DataExchange`.

[7] Apache ActiveMQ, `http://activemq.apache.org`.

[8] M. Sikorski, Z. Jiang, M. Sprung, S. Narayanan, A. R. Sandy and B. Tieman, "A graphical user interface for real-time analysis of XPCS using HPC," Nucl. Instrum. Method A 649(1), (2011) 234.

[9] B. Tieman, S. Narayanan, A. Sandy and M. Sikorski, "MPICorrelator: A parallel code for performing time correlations," Nucl. Instrum. Method A 649(1), (2011) 240.

ISBN 978-3-95450-139-7