# Data Retrieval and Parsing of Form 4 from the Edgar System using Multiple CPUs

Raja H. Singh*, Nolan Burfield*, Frederick Harris, Jr.*
*Department of Computer Science and Engineering, University of Nevada, Reno
email: rajas@nevada.unr.edu, nburf@nevada.unr.edu, Fred.Harris@cse.unr.edu

***Abstract*—In this paper we present a parallelized system that retreives and parses Form 4 documents from the Securities and Exchange Commission's Electronic Data Gathering, Analysis and Retrieval database (EDGAR). This information is very important for investors looking at insider trading information to make investment decisions. However, the information's usefullness is inversly related to the time it takes to retreive and analyize the information. A sequential system is slow due to the latency associated with the retreval and parsing of the Form 4s, which on average exceeds 1000 per day. By parallelizing the retreival and parsing of Form 4s we were able to attain the max speed up of 20x, resulting in parsing of a daily index with 1000 forms in under 30 minutes instead of 9 hours it takes utilizing a single processor.**

***Index Terms*—Commodity Cluster; Sun Grid Engine; Investment; Open MPI; Java; MySQL; XML Parsing; Document Object Model;**

## I. Introduction

The Securities and Exchange Commission (SEC) is the regulatory branch of the government that is responsible for "protecting investors, maintaining fair, orderly, and efficient markets, and facilitate capital formation [1]." According to an article on Gallup [2] over 54 percent of Americans own stocks. SEC states that individuals investing in the stock market do so to fund goals such as paying for their homes, sending their children to school, and to secure their futures. Unlike banking, stock investments are not guaranteed by the Federal government and have a high potential of losses. Due to the nature of the investments, the SEC recommends conducting sufficient research about the company prior to investing in their stocks.

The SEC's primary objective is to provide investors with the opportunity to research the company/companies prior to investment. To carry out the said objective, the SEC requires public companies to disclose "meaningful financial and other information to the public [1]." One such form (Form 4) is part of the required disclosure.

### A. Form 4: Insider Trading

Forms 4, 4A, 3, 3A, 5 and 5A are categorized as Insider Trading forms. Contrary to the belief that insider trading is illegal, SEC specifies that it is used for both illegal and legal conduct [3]. In the context of this paper we will refer to insider trading as purchasing or selling of stocks in a company by individuals where they are corporate officers. Any corporate insider (company's officers, directors, or beneficial owners who have more than 10 percent of a company's equity) are required to file a "statement of ownership regarding those securities [3]" using Form 3. Form 4 is used to report the changes in ownership i.e. purchasing and selling shares. Form 5 is used to report any transactions that the insider didn't report on form 4; Form 5 is used to report any transactions that should have been reported on Form 4 [3].

According to [4], Form 4 is the most filed form between the years 1994 - 2011, with more than 4 million filings according to the authors. This also holds true today, for example in the year 2015 there were 397910 Form 4 filings, with 82126 of those Form 4 filings in the fourth quarter of 2015 alone. In order to parse the large number of these forms, a system has to be created to do so automatically at a reasonable speed so that the useful nature of insider trading information is preserved.

### B. Form 4: Potential Uses

We will focus on the retrieval of Form 4. The reason behind this is that several investors look to Form 4 to determine the change of ownership within the company. This is vital information because it indicates the confidence in the company's growth potential. For example, an investor is able to look at Form 4 filed by a high ranking officer and determine if the insider is accumulating or disposing of their holdings in the company. An investor may utilize this information when considering the purchase or sale of shares. For example, if the insider is purchasing shares, it could possibly indicate that they have internal information indicating potential growth of the company. However if an insider disposes shares in the company then it indicates their knowledge of potential downturn in the company's performance in the future. In this paper we will provide an overview of the system, and its optimization through the use of multiple CPUs.

## II. Prior Works

The financial data vending industry is largely corporate, therefore there have not been many published works on agents downloading and parsing information from the SEC. Up to the point of writing this paper, we were only able to locate two papers that discussed the retrieval of data from the EDGAR system. [4] by Garcia and Norli offer a method to crawl EDGAR for Form 8k, the second most common filed form after Form 4, is used to announce major events to shareholders. However, Garcia and Norli utilize it to analyze CEO turnovers.

The writers wrote a Perl program to download all 8k files locally and then parse through and analyze the forms.

The second paper [5], written by D. A. Lyon, discusses parsing the Central Index Keys (CIKs) and correlating them to stock tickers. Currently EDGAR only uses company name or CIK to present the data on the requested company. Lyon created a graphical user interface that allows the user to enter the company ticker to gather the data from EDGAR; Lyon used Java to write HTML scrapers to gather data from the EDGAR System. We were not able to locate any papers addressing gathering and parsing Form 4, however we did discover companies that charge a monthly subscription fee to disseminate Form 4 information to their subscribers.

## III. Data Source & Format

SEC maintains all filings on their Electronic Data Gathering, Analysis and Retrieval database (EDGAR) system. The data can be accessed either through their EDGAR search page or through SEC's File Transfer Protocol (FTP) server. The formats offered by the SEC are HTML, "eXtensible Business Reporting Language" (XBRL) and Extensible Markup Language (XML). Our system pulled the Form 4 pages in XML format. Please refer to Figure 1 for the format of the document.

### A. File Transfer Protocol (FTP) server

The FTP server allows users to log into the system anonymously by utilizing your email address. We utilized this FTP method, making anonymous FTP calls to the server. Refer to [6] for specific instructions on logging into the system. The SEC does request bulk ftp requests be performed between 9pm and 6am ET.

*1) Indexes:* The EDGAR system provides indexes for FTP retrievals. These indexes list Company Name, Form Type, CIK, Date Filed and the File Name; this index also includes the folder path to the document [6]. Please refer to Figure 2 for the image of the EDGAR indexes set-up. The system offers four types of indexes:

- company -Index file sorted by the company name.
- form - Index file sorted by form type.
- master-Index file sorted by the CIK.
- XBRL - "List of submissions containing XBRL financial files, sorted by CIK number; these include Voluntary Filer Program submissions [6]."

The indexes are filed daily on the EDGAR system containing all submissions made to the organization. The system also maintains a quarterly index list containing all filings for the quarter for a given year. These date back to 1994 as shown in Figure 2. Inside each index, as stated above, is the listing of all files based on the type of index file chosen (i.e. company vs form vs master). Please refer to Figure 3 of the image of a daily index file (type master).

## IV. System Overview

The system is built to pull all data from the SEC and store it into a MySQL database. It relies on multiple classes that can be categorized into specific functionality of the system.


Figure 1. This shows the XML format of the downloaded Form 4 to be parsed by the system.

Due to the space limitations we will not discuss every class within the system but will instead discuss classes that assist us in giving a general overview the system in the following sections.

### A. FTP

The system relies heavily on the FTP class. The primary function of this class is to open an FTP connection, and connect to a specified destination. The connection stays open until either it is closed or the system exits. This class must be instantiated when used in order to keep a persistent connection and handle connection failures. Running the parallel application several nodes may fail when attempting to pull FTP data, therefore the FTP class must handle this and still grab the necessary data. The approach the system takes is to continue to attempt a connection until the data is pulled. To avoid corruption in the data FTP is set to transfer as binary, which is not the default for the Java Apache Commons-net FTPClient [7].

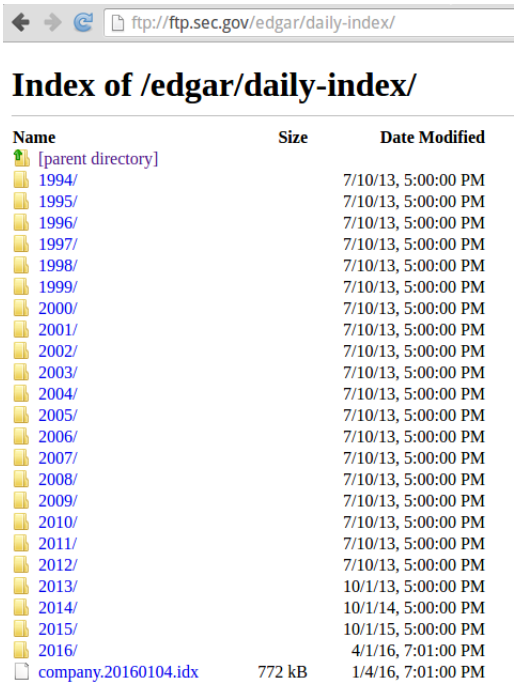


Figure 2. This figure shows the EDGAR file-path with all the daily indexes dating back to 1994.

### B. System updating

The system initializes by checking the last date the MySQL database was updated and computes the number of days that it needs to retrieve and store from the EDGAR system. To achieve this we created a set of nested classes that compute the date difference between the current time and the last known update date for the database. Since the SEC filings are only accepted and updated Monday through Friday, the system does not include the weekends when trying to retrieve data from the system.

### C. Index Retrieval

The EDGAR system has an efficient method to disperse information on filings for specific dates as explained in prior sections that rely on the "index schema." Our system uses the FTP connections to pull the index for a specified time frame (either daily index for a specific date or a quarterly index for a specific quarter of a year). The classes responsible for this retrieve the index and then parse the index storing each filing into a HashMap, where the key is the type of the filing (ie. Form 4) and the value is an ArrayList of a type DailyData. Please refer to [8] for details about HashMap and ArrayList data structures. The Daily Data is a container that stores every component of the index information listed below:

- Company Name
- Form Type
- CIK
- Date Filed
- File Name
- Folder path to the document

Once the data from the daily index file is parsed, and stored into the appropriate data structure, the information is passed to each form parsing class. Meaning the index retrieval class iterates through the HashMap and invokes the appropriate class to handle each type of the form. As of now our system only has parses for the insider trading forms. The system is designed for easily adding in new SEC form parsers without the need to modify other classes.

### D. Data Retrieval

The system currently has a class to retrieve the list of forms passed to it by the index retrieval class. For example when the Form 4 class is invoked it is passed a list of DailyData objects that contain the location of the forms in the EDGAR system. The class iterates through the list and makes FTP requests to pull the specified form one at at time and parses the retrieved data.

### E. Data Parsing Class

To parse the XML data retrieved from Edgar, we created the Form 4 parsing class. The class builds a Document Object Model (DOM) tree object and recursively iterates through all of the nodes within the tree storing the values into a string builder object. In order for us to parse the retrieved documents, we had to know the structure (values) of the XML document prior to computing. We utilized the insider trading form specifications provided by the SEC in [9] to create the parsing class. If the element/node was not found, a default value of null is stored for the field.

For the purpose of testing the efficiency of parallelization of the bottleneck of the system, we modified the code so that the values were dumped into a CSV file instead of storing it in a database. We decided to not store the values into a database for this paper because the parallel execution of the system. In order to store the data into a MySQL database the system requires a pool of Java Database Connectivity(JDBC) connections, unfortunately it was nonsensical on the Sun Grid Engine (SGE) using MPI. If we had chosen to proceed with the storage of the data into MySQL database, it would have resulted in large network overhead. This is irrational due to the pre-existing network caused latency.

**Refer to Algorithm 1 for the pseudo algorithm of the system.**

## V. Hardware and Language

The code was written in Java programming language due to the cross platform nature of Java and the vast number of libraries available for software development. Apache Ant was used to build/compile the system and was executed on the Oracle Grid Engine (previously known as the Sun Grid Engine). Message Passing Interface (MPI) was used to handle the communication between the CPUs in the cluster.

The grid has 27 Dell Servers with 16 cores each for a total of 432 cores. Each server has the following specification [10]:

- Dual 8 core Intel E2650v2 2.6 GHz processors
- 256 GB RAM

```
Description:          Daily Index of EDGAR Dissemination Feed by Company Name
Last Data Received:   Apr  8, 2016
Comments:             webmaster@sec.gov
Anonymous FTP:        ftp://ftp.sec.gov/edgar/



Company Name                                                  Form Type   CIK         Date Filed  File Name
---------------------------------------------------------------------------------------------------------------------------------------------------------------
1347 Property Insurance Holdings, Inc.                        8-K         1591890     20160408    edgar/data/1591890/0000914760-16-000397.txt
1st Century Bancshares, Inc.                                  PREM14A     1420525     20160408    edgar/data/1420525/0001047469-16-012053.txt
56 Brewing, LLC                                               D           1671634     20160408    edgar/data/1671634/0001671634-16-000001.txt
6D Global Technologies, Inc                                   8-K         1382219     20160408    edgar/data/1382219/0001185185-16-004185.txt
86Borders LLC                                                 D/A         1622720     20160408    edgar/data/1622720/0001622720-16-000004.txt
A-KNH-33-Fund, a series of AngelList-SDA-Funds, LLC           D           1671392     20160408    edgar/data/1671392/0001671392-16-000001.txt
A-MJQ-13-Fund, a series of AngelList-BaRs-Funds, LLC          D           1669974     20160408    edgar/data/1669974/0001669974-16-000001.txt
A-NHN-34-Fund, a series of AngelList-S2Al-Funds, LLC          D           1671393     20160408    edgar/data/1671393/0001671393-16-000001.txt
A-PKY-15-Fund, a series of AngelList-MaBr-Funds, LLC          D           1671391     20160408    edgar/data/1671391/0001671391-16-000001.txt
A-VRO-11-Fund, a series of AX-IA-Funds, LLC                   D           1671390     20160408    edgar/data/1671390/0001671390-16-000001.txt
A10 Networks, Inc.                                            3           1580808     20160408    edgar/data/1580808/0001209191-16-113760.txt
AB CAP FUND, INC.                                             DEF 14C     81443       20160408    edgar/data/81443/0001193125-16-534930.txt
AB CAP FUND, INC.                                             DEFA14C     81443       20160408    edgar/data/81443/0001193125-16-534946.txt
ABERCROMBIE & FITCH CO /DE/                                   8-K         1018840     20160408    edgar/data/1018840/0001193125-16-535655.txt
ABIOMED INC                                                   8-K         815094      20160408    edgar/data/815094/0001193125-16-535148.txt
ABNER HERRMAN & BROCK LLC                                     13F-HR      1038661     20160408    edgar/data/1038661/0001038661-16-000007.txt
ABRAMSON STEVEN V                                             4           1232980     20160408    edgar/data/1232980/0001005284-16-000182.txt
ACADIA REALTY TRUST                                           8-K         899629      20160408    edgar/data/899629/0001144204-16-093488.txt
ACCELERON PHARMA INC                                          4           1280600     20160408    edgar/data/1280600/0001179110-16-022918.txt
ACCELERON PHARMA INC                                          4           1280600     20160408    edgar/data/1280600/0001179110-16-022919.txt
ACHILLION PHARMACEUTICALS INC                                 8-K         1070336     20160408    edgar/data/1070336/0001193125-16-535465.txt
ACORN COMPOSITE Corp                                          4           1518311     20160408    edgar/data/1518311/0001078782-16-002595.txt
```

Figure 3. This figure shows contents of a daily index file. It is an index of all the filings for the specified date, including the name of the filing entity, the type of form filed, the CIK for the filer and the location (file-path) of the form.

**Algorithm 1** System Overview

```
Compute the date difference between current date and last database
update date  initialize retrieval of data for each day
for each date retrieve the daily index do
    for each retrieved Daily index file: initialize index parsing do
        for each parsed index file do
            Store each form information into a Daily Data object
            Store each Daily Data object into a
          Hash Map <Key:Form Type, Value: List<Daily Data>>
            Invoke the appropriate parsing class to parse and
                                        store data into database.
        end for
    end for
end for
```

- 10 GBPS Ethernet
- 1.2 TB of local storage

## VI. Profiling: Parallel Justification

Prior to profiling the code we already knew that the system's latency came from the ftp requests, however it was required to prove this in order to justify the parallelization of the system. We did not use the HPROF profiler because of the complications associated with submitting jobs to the Oracle Grid Engine.

In order to profile the sequential execution, we created a profiling class using the `System.nanoTime()`. Refer to [11] for detailed specifications on the Java system timer. Using the profiling class, we were able to time every function call made by the system. Table I shows the three of the most time intensive functions in the execution of the program. Please note that the main function calls the RunForms function which entails calls the FTPRequests function. Therefore the times for the calling functions exceed the times of the function(s) being called. Based on the timings shown in Table I we were able to determine the FTPRequest function was the appropriate function to parallelize because it contributes (on average) 94.5% to the total computational time.

Table I
THE TABLE ABOVE SHOWS THE MOST TIME INTENSIVE FUNCTION CALLS.

| | 50 | 75 | 100 | 250 | 500 | 750 | 1000 |
|---|---|---|---|---|---|---|---|
| main | 27.66 | 41.06 | 54.52 | 135.80 | 269.36 | 405.79 | 541.09 |
| RunForms | 27.01 | 40.50 | 53.98 | 135.26 | 268.81 | 405.24 | 540.54 |
| FTPRequest | 26.12 | 38.93 | 51.68 | 128.43 | 256.06 | 384.51 | 512.10 |

## VII. Sequential Execution

The sequential code is described in Section IV System Overview and Algorithm 1. The sequential code was also executed on the Oracle Grid Engine to ensure consistency. For the parsing we utilized a single daily index with 1025 Form 4 documents in the index list. We did a total of 7 iterations, increasing the number of Form 4 documents parsed during each iteration; we parsed the following number of forms for each iteration 50, 75, 100, 250, 500, 750 and finally 1000. This allowed us to compare the sequential and parallel executions.

## VIII. Parallel Execution

For the parallel execution every computer node (CPU) opens the daily master index shown in Figure 3, parses the index list as mentioned in Algorithm‘1, and stores it into a HashMap. Next each node strides the Form 4 list extracting DailyData objects for each Form 4 that node is responsible for. The nodes stride the ArrayList of DailyData types based on their rank and the total number of nodes initialized on the grid. Refer to Algorithm 2 for an overview of the logic for each node in the parallel execution.

Once each node creates a sub list of its Form 4 documents, the rest of the code each node executes is the same as sequential. Due to Java's garbage collection feature, utilizing

**Algorithm 2** Parallel System Overview

```
Retrieve the daily index
for each retrieved Daily index file: initialize index parsing do
    for each parsed index file do
        Store each form information into a Daily Data object
        Store each Daily Data object into a
         Hash Map <Key:Form Type, Value: List<Daily Data>>
        for each Form Type do
            Stride the List<Daily Data>
            Build new list based off of rank
            Invoke the appropriate parsing class to parse
            Output parsed data to CSV
        end for
    end for
end for
```

shared memory was not feasible. Therefore each node ran the same code in parallel without communication with any other nodes. The sequential algorithm we parallelized in this manner is the ideal "embarrassingly parallel problem."

### A. Set-up

As with the sequential execution, there were a total of 7 iterations with each iteration increasing the number of forms as follows: 50, 75, 100, 250, 500, 750, and 1000. However, for each one of the iterations where the number of forms was increased, the parallel execution ran each iteration five times starting with 4 nodes and increasing by 4 up to a total of 20 nodes; the number of nodes used for each form iteration were 4, 8, 12, 16, and 20.

To avoid the networking bottleneck when making FTP calls, we utilized a specific round-robin scheduling scheme to choose workers. The system was initialized so that the slots (number of nodes requested) "are filled one per machine until all machines have one slot filled, and then the next slot on each machine is filled, until all requested slots are provided [10]." However since there are 29 servers and the max number of nodes we requested are 20 at any given time, we never ran into a scenario with more than one node per server/box.

## IX. Results

The results were as expected. The sequential execution of the data retrieval and parsing takes longer than reasonable times. An average daily index contains over 1000 Form 4 documents. At the current execution speeds, a sequential execution of the system takes around 9 hours to parse 1000 forms. This is unacceptable in the financial industries where time is very critical factor when determining how to react on the information attained; the benefit of any information is inversely proportionate to the time. The longer the information is out there, the less it is worth. Once the system was parallelized the runtime drops drastically as shown in Table II. For example, parsing 1000 forms with 4 CPUs resulted in a completion time of around 2 hours and 30 minutes. Figure 4 visualizes this and can show that by simply adding in 4 nodes will result in an acceptable data collection time (Figure 5 shows this data with normalized values).

Since the bottleneck of the system could be divided into individual parts and executed independently of each other, the speedup increased in conjunction with the number of processors used. This resulted in a relatively constant speedup for each node size regardless of the number of forms parsed. For example using 4 processors the speedup stayed between 3.5X to 4X. This makes sense, because even though there was no communication between the processors, they were still vying for the same network resources, resulting in a lag. Table III shows that values of speedup through all iterations, as the number of forms parsed increases the speedup increases as well. However, speedup of all nodes does not change drastically. With 20 nodes speedup is about 20 when 1000 forms are parsed. Please refer to Figure 6 for the visualization of the speedup values; it shows the gradual increase of speedup for each number of nodes.

The key function of the system is to retrieve and extract EDGAR data and store it into local database. The speed at which the data is extracted is a very important metric to determine whether the system is feasible for use in the real world scenario. Currently our system (upon parallelization) parses around 34 forms per minute using 20 nodes. Table IV shows the values for each iteration performed, and these values are visualized in Figure 7. Looking at the trends in the graph it can be seen that the throughput follows the trend of speedup exactly.

Table II
THE TABLE ABOVE SHOWS THE RUN TIMES OF THE PROGRAM THROUGHOUT EACH PROCESSOR SIZE AT EACH ITERATION OF QUANTITY OF PARSED FORM 4.

| | 1 | 4 | 8 | 12 | 16 | 20 |
|---|---|---|---|---|---|---|
| 50 | 27.66 | 7.55 | 4.33 | 3.25 | 2.73 | 3.26 |
| 75 | 41.06 | 10.94 | 5.95 | 4.33 | 3.27 | 2.74 |
| 100 | 54.52 | 14.05 | 8.08 | 5.40 | 4.53 | 3.73 |
| 250 | 135.80 | 34.46 | 17.97 | 11.85 | 9.93 | 7.97 |
| 500 | 269.36 | 68.11 | 34.63 | 23.16 | 19.85 | 14.46 |
| 750 | 405.79 | 102.05 | 51.12 | 35.01 | 28.56 | 22.13 |
| 1000 | 541.09 | 135.54 | 68.16 | 45.77 | 37.15 | 28.75 |

Table III
THE TABLE ABOVE SHOWS THE SPEEDUP OF THE PROGRAM THROUGHOUT EACH PROCESSOR SIZE AT EACH ITERATION OF QUANTITY OF PARSED FORM 4.

| | 1 | 4 | 8 | 12 | 16 | 20 |
|---|---|---|---|---|---|---|
| 50 | 1.00 | 3.66 | 6.38 | 8.50 | 10.11 | 8.46 |
| 75 | 1.00 | 3.75 | 6.89 | 9.46 | 12.53 | 14.97 |
| 100 | 1.00 | 3.87 | 6.74 | 10.09 | 12.02 | 14.58 |
| 250 | 1.00 | 3.94 | 7.55 | 11.45 | 13.67 | 17.01 |
| 500 | 1.00 | 3.95 | 7.77 | 11.62 | 13.56 | 18.61 |
| 750 | 1.00 | 3.97 | 7.93 | 11.58 | 14.20 | 18.33 |
| 1000 | 1.00 | 3.99 | 7.93 | 11.81 | 14.56 | 18.81 |

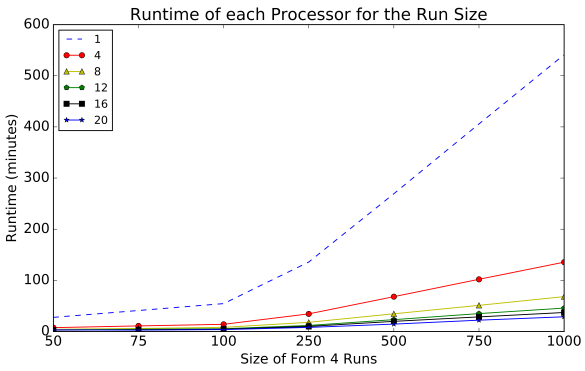


Figure 4. The total time in minutes to finish execution of each processor size at each iteration of quantity of parsed Form 4.

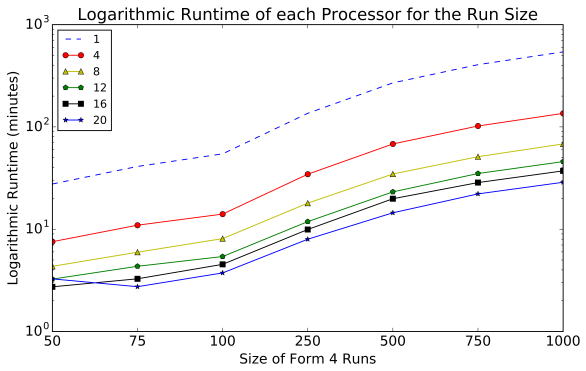


Figure 5. This shows that same as Figure 4 with the data scaled to log 10 along the time axis to better show run time differences.

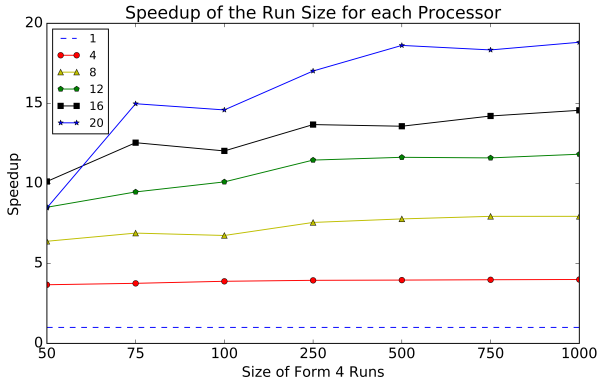


Figure 6. This figure shows the speedup of the program run time, throughout each processor size at each iteration of quantity of parsed Form 4, with the baseline being the sequential execution time (Sequential Run Time / Parallel Run Time).

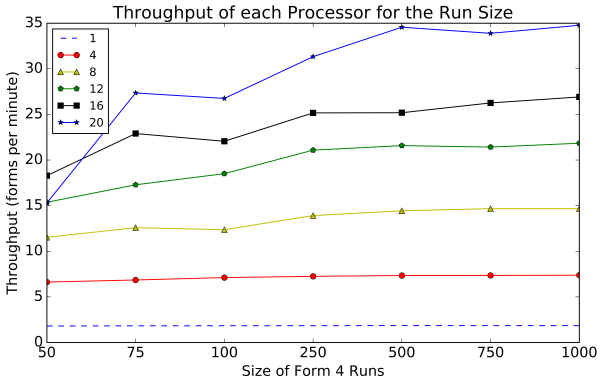


Figure 7. This figure shows the throughput of the Form 4 parsing, the system throughput is calculated on forms parsed per minute.

## X. Conclusion and Future Work

As stated in the problem, it is necessary with financial data to receive the information as soon as possible in order to react to the changing markets. In order to do this a delay of 9 hours would not be an acceptable time frame to get information from a Form 4 document filed in the Edgar database. Parallelization of the program successfully dropped that time delay down to only 27 minutes. This time was achieved with running the same algorithm on 20 processors at once, and since the nodes all run the same logic there is no need for communication between them.

Achieving a large speedup was the purpose of the parallelization of the bottleneck within the program. That is to

Table IV
THE TABLE ABOVE SHOWS THE THROUGHPUT OF THE PROGRAM THROUGHOUT EACH PROCESSOR SIZE AT EACH ITERATION OF QUANTITY OF PARSED FORM 4.

| | 1 | 4 | 8 | 12 | 16 | 20 |
|---|---|---|---|---|---|---|
| 50 | 1.80 | 6.62 | 11.53 | 15.37 | 18.28 | 15.29 |
| 75 | 1.82 | 6.85 | 12.58 | 17.28 | 22.90 | 27.35 |
| 100 | 1.83 | 7.11 | 12.36 | 18.50 | 22.05 | 26.75 |
| 250 | 1.84 | 7.25 | 13.91 | 21.08 | 25.16 | 31.33 |
| 500 | 1.85 | 7.34 | 14.43 | 21.58 | 25.18 | 34.56 |
| 750 | 1.84 | 7.34 | 14.67 | 21.42 | 26.25 | 33.88 |
| 1000 | 1.84 | 7.37 | 14.67 | 21.84 | 26.91 | 34.77 |

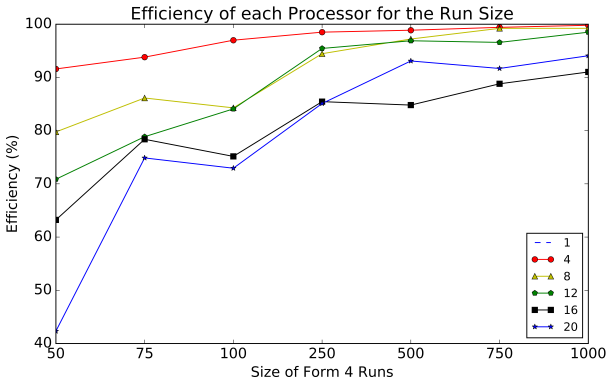


Figure 8. Efficiency.

say processing as many forms per minute as possible is most beneficial when talking about the financial data. Since an average daily size of Form 4 submissions is around 1000 and these can be fully processed and stored in under 30 minutes the parallel program accomplishes the goals set out to be done.

There is much work that can be done to make the system a comprehensive tool to gather the data from EDGAR. Please see the list below for future work:

- Add parsing class to parse all key forms such as 10-k, 10-Q, 8-K, 13-F, 13-H, etc.
- Add a View to the system that visualizes the data both in a tabular and graphical manner.
- Add a package to the sytem that extracts historical data so back testing can be performed to check the validity of any trading strategies.
- Create a class that utilizes Map Reduce to parse complex forms with XBRL formats.
- Write a RSS (Rich Site Summary) reader to update the system live based on the up to date filing information.

Form 4 information offers great a great indicator of company's direction, performance/profit wise. The system can be utilized as stand alone for long term investments or it could be used in conjunction with the historical data. For example the system could be utilized to find insiders with a track record of indicating the future behavior of a company. This can be done by back testing their past insider trading actions and comparing them to the historical movement of the company. If they have a track record of making appropriate trades based on possible future movements that specific insider's actions can be tracked to make investment decisions. The possible uses of the insider trading are too many to list them all in this paper.

## ACKNOWLEDGMENT

This material is based in part upon work supported by the National Science Foundation under grant no. IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

[1] "What We Do." SEC.gov. Web. 05 Apr. 2016.

[2] "In U.S., 54% Have Stock Market Investments, Lowest Since 1999." Gallup.com. N.p., n.d. Web. 14 Dec. 2015. <http://www.gallup.com/poll/147206/stock-market-investments-lowest-1999.aspx>.

[3] "Fast Answers." SEC.gov. Securities and Exchange Commission, n.d. Web. 2 Mar. 2016. <https://www.sec.gov/answers/form345.htm>. Forms 3, 4, 5

[4] D. Garca and . Norli, Crawling EDGAR, The Spanish Review of Financial Economics, vol. 10, no. 1, pp. 110, Mar. 2012.

[5] D. A. Lyon, Multi-threaded data mining of EDGAR CIKs (Central Index Keys) from ticker symbols, 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008.

[6] "Information for FTP Users." Information for FTP Users. Securities and Exchange Commission. Web. 5 Mar. 2016. <https://www.sec.gov/edgar/searchedgar/ftpusers.htm>.

[7] "Apache Commons Net Overview." Apache Commons Net Overview. Apache, n.d. Web. 21 Mar. 2016. <https://commons.apache.org/proper/commons-net/>.

[8] "The Java Tutorials." The Java Tutorials. Oracle, n.d. Web. 27 Jan. 2016. <https://docs.oracle.com/javase/tutorial/>.

[9] "EDGAR Ownership XML Technical Specification (Version 5.1)." EDGAR Ownership XML Technical Specification (Version 5.1). Securities and Exchange Commission, n.d. Web. 15 Mar. 2016. <https://www.sec.gov/info/edgar/ownershipxmltechspec.htm>.

[10] "High Performance Computing (The Grid)." The Grid Specifications. Web. 15 Mar. 2016. <http://www.unr.edu/it/research-resources/the-grid>.

[11] "Java Platform SE 8." Java Platform SE 8. Oracle. Web. 17 Mar. 2016. <http://docs.oracle.com/javase/8/docs/api/>.

[12] "Using EDGAR - Researching Public Companies." Investor.gov. Securities and Exchange Commission, n.d. Web. 1 Feb. 2016. <https://www.investor.gov/researching-managing-investments/researching-investments/using-edgar-researching-public-companies>.

[13] Gupta, Lokesh. "Java XML DOM Parser Example Tutorial." HowToDoInJava. N.p., 31 July 2014. Web. 10 Feb. 2016. <http://howtodoinjava.com/xml/java-xml-dom-parser-example-tutorial/>.

[14] "Welcome - Apache Ant." Apache Ant. N.p., n.d. Web. 4 Apr. 2016. <http://ant.apache.org/>.

[15] "About JDBC Resources and Connection Pools." (Sun Java System Application Server Platform Edition 8.2 Administration Guide). Oracle, n.d. Web. 24 Feb. 2016. <https://docs.oracle.com/cd/E19830-01/819-4712/ablii/index.html>.

[16] "MPI Documents." MPI Documents. Web. 12 Mar. 2016. <http://www.mpi-forum.org/docs/>.

[17] "Chapter 4 Tutorial." MySQL. N.p., n.d. Web. 29 Jan. 2016. <http://dev.mysql.com/doc/refman/5.7/en/tutorial.html>.