# EFFICIENT FEATURE RICH DATASETS FOR DATA SCIENCE ALGORITHMS

Karthik Pai B H[1], Sampath Kini K[2]
[1]Associate Prof, Dept of Information science and Engineering, NMAMIT, Nitte, Karkala, India,
[2]Assistant Prof, Dept of Computer science and Engineering, NMAMIT, Nitte, Karkala, India

**Abstract**
**One of the key data structures used in data science and mining algorithms is datasets of numerical elements. Traditional datasets represent a collection of data items that support read/write operations of data elements. Data science and mining algorithms use these data structures for processing data elements in the memory. It is observed that aggregate operations such as max, min, sum, average, and variance are quite often used by most of the data mining algorithms. Programming languages come with library of variety of data structures to represent collection of data elements. For example, Java supports collection classes such as ArrayList, HashMap to represent collection of elements in the memory. However, these collections classes do not provide in-built aggregate operations on data elements. This research paper explains the efficient techniques that can be used to perform aggregate operations on datasets using multithreading/multicore technologies. This research paper also compares elapsed time determined for aggregate operations with and without multithreading options for varying sizes of datasets**
**Index Terms: Dataset, data science, Aggregate functions, data mining, multithreading, multicore**

## I. INTRODUCTION

Most of the data mining algorithms apply aggregate methods on collection of data elements. Aggregate methods are typically used to summarize the statistical analysis. For example, maximum sales of a product or average score of sports person. Programming languages such as Java comes with library of collection classes to represent container of data elements. These classes provide methods for inserting, deleting and reading of elements. Since these classes do not have methods for performing aggregate operations such as max, min, sum, average, programmers themselves need to write software code to determine aggregate values of datasets by sequentially going over dataset elements.

This research paper captures efficient technique used for performing aggregate operations on elements of dataset. Traditional technique is to run through all elements of dataset sequentially in a single thread. Enhanced technique is to split the dataset into multiple partitions, perform aggregate operation on each partition on multiple cores simultaneously and finally combine the results obtained from each partition. Java's Executor service framework is used for performing aggregate operation of a partition on a core. Additionally, this enhanced dataset implementation provides a mechanism to cache already determined aggregate values of a dataset elements. This caching is achieved by this enhanced dataset with the usage of placeholder variables for already computed aggregate values. These partitioning, multithreading and caching techniques will help to gain performance improvements when aggregate methods are invoked by data mining algorithms. Experiments were conducted for aggregate methods max, average, variance on a dataset of varying sizes.

The objective of this paper is implemented in two stages. 1) Determining elapsed time with single and with multiple threads. 2) Presentation and

analysis of elapsed time for all the approaches. Elapsed time is recorded after performing aggregate operations on a collection of data elements. Thereafter, recorded values are used to present the analysis.

This paper is organized as follows. In section II, Details of master slave approach is described. In section III, dynamics of master slave design is captured. In section IV, Usage of master slave design for performing aggregate operations on multiple cores. In section V, results in terms of elapsed time taken by aggregate operations are presented. In section VI, conclusions about aggregate operations on datasets using multicores are drawn.

## II. MASTER SLAVE DESIGN

As described in [1, 2], A master element splits the job into various similar slave elements. It computes a final result from the results obtained from these slaves. Figure (1) shows class structure of the master slave design.

**Master element:**

- It applies the 'divide and conquer' principle for job split.
- It implements functions for job split into various equal sub tasks.
- It starts and controls processing and computing a final result from all the results from slaves.
- It also manages references to all slaves instances to which it submits the computation of subtasks.

**Slave component:**

- Exposes a sub-service that can compute the subtasks delegated by the master
- There will be at least two instances of the slave elements connected to the master

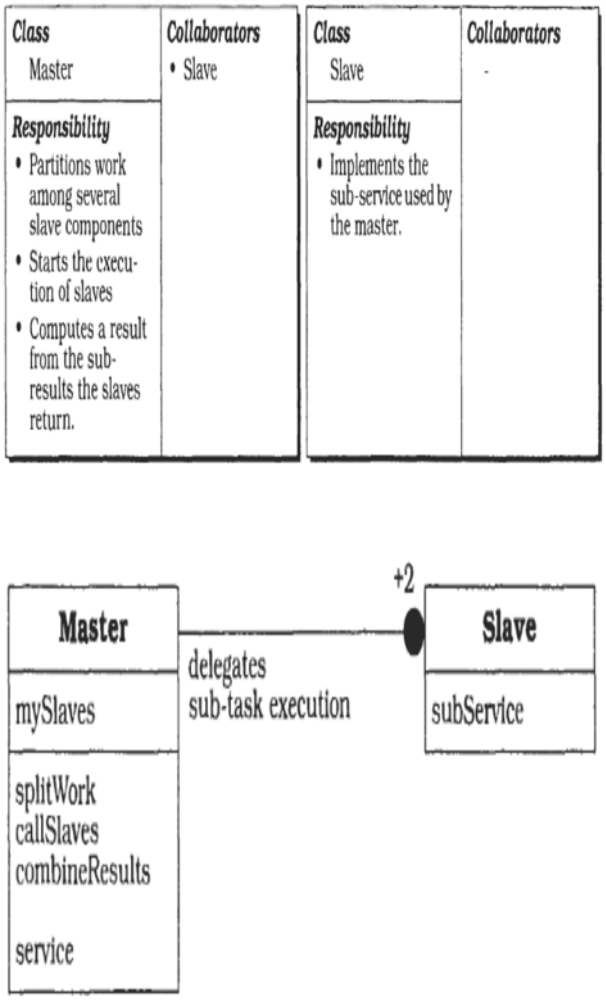


Figure 1. master-slave design with help of a class diagram

## III. DYNAMICS OF MASTER SLAVE DESIGN

Figure (2) shows object interaction among client, master and slaves during runtime operation. As described in [1, 2], The scenario comprises six phases:

- A client puts a request for the master.
- The master does job split into equal various sub-tasks.
- The master delegates the computation of sub-tasks to various slaves, initiates their execution and waits for their results.
- The slaves execute sub-tasks and return the results of their execution to the master.
- The master determines a final result for the entire job from the sub results obtained from the slaves.
- The master forwards this result back to the client.

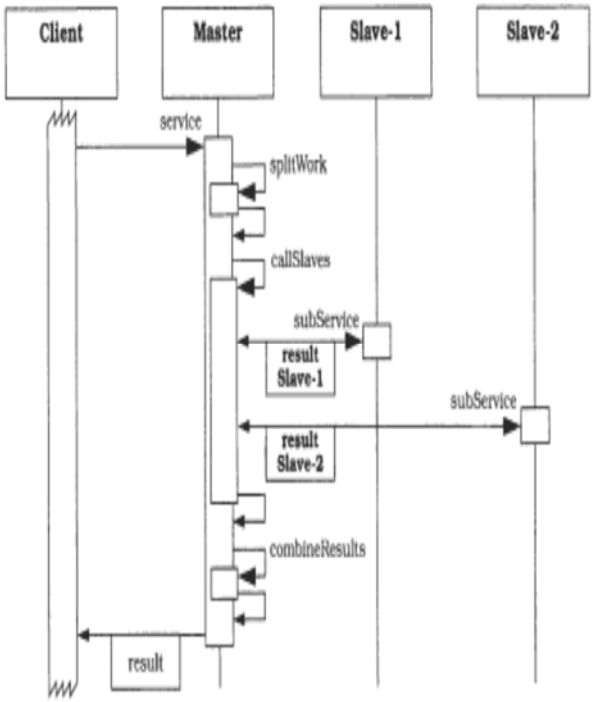


Figure 2. Object interaction diagram showing a runtime scenario of master-slave design

## IV. USAGE OF MASTER SLAVE DESIGN FOR PERFORMING AGGREGATE FUNCTIONS

The collection class supporting aggregate methods will play the role of master. In this case study, the java collection class ArrayList is used to represent dataset. The client will invoke aggregate method implemented as part of dataset implementation. This will trigger job split where in entire dataset is partitioned into number of cores available in the computing system. Each partition is represented by the slave class callableArrayList. Each instance of this class will be running on individual core assigned by Executer service class. That means, each instance will be computing aggregation on its own dataset partition assigned by master. Finally, the master will combine the results returned by each of the slave instances; it caches the result in instance variables and sends the result back to client. Figure (3) shows java classes developed using master slave design approach. The class AggrArrayList will act as master class. The class ArraListCallable will act as slave class. This slave class does implement callable interface for it to be running on individual core assigned by the class ExecuterService. Figure (4) shows object interaction among client, AggrArrayList the master and object instances of ArrayListCallable the slaves.

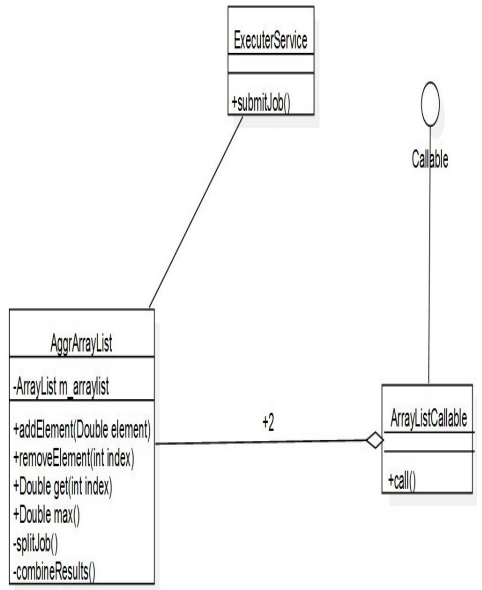


Figure 3. Class diagram showing the java classes designed for experiments

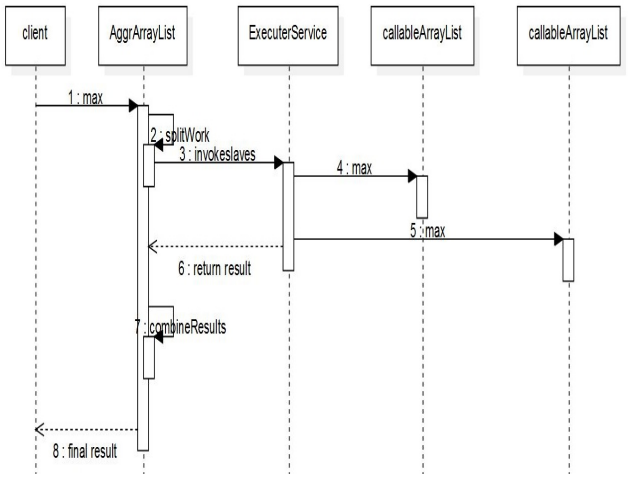


Figure 4. Object interaction diagram showing a scenario of performing max aggregate operation

## V. RESULTS OF EXPERIMENTS

Observed and recorded elapsed time for all the runs is shown via graph.(Figure 4,5,6).These experiments were conducted on a data set of numerical elements of type Double in Java programming language. Varying workload unit (number of elements) is as shown in the graph. Experiments were conducted on following system environment.

- Processor: Intel(R) Pentium(R) CPU N3540 @2.16GHZ
- RAM : 4GB
- No: of cores: 4
- System type: windows 8.1, 64 bit, x-64 based processor
- Java/JVM: 1.8

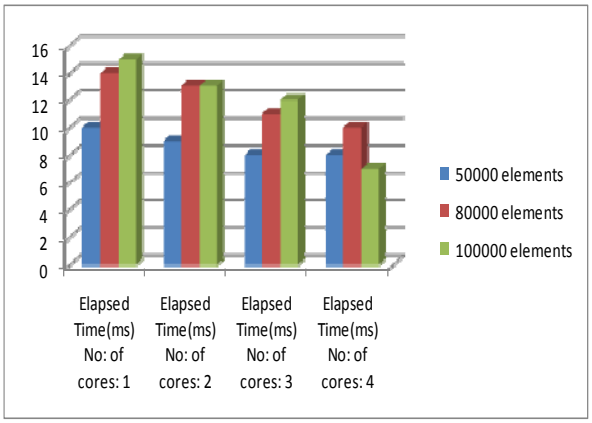


Figure 5. . Elapsed time determined with multicores on max aggregate operation

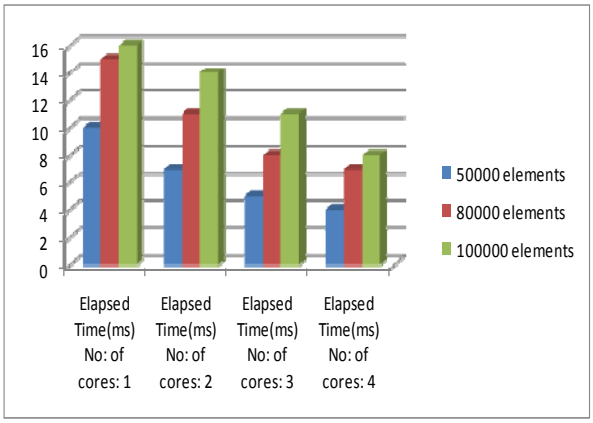


Figure 6. Elapsed time determined with multicores on avg aggregate operation

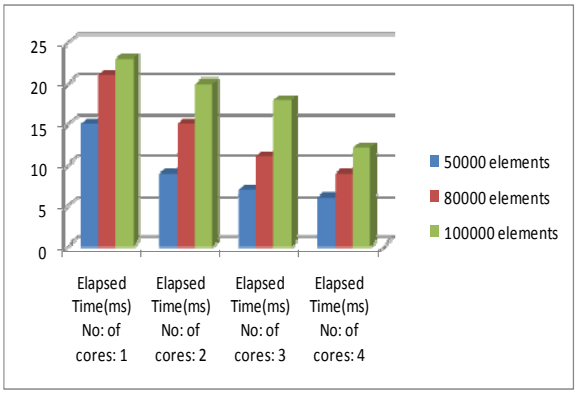


Figure 7. Elapsed time determined with multicores on variance aggregate operation

## VI. Conclusions

- This research provides experimental findings of aggregate operations on datasets using multithreading concept.
- Elapsed time is determined with varying size of datasets on multicores for aggregate operations.
- Comparison in terms of elapsed time across multiple threads is drawn.
- Provides feature rich efficient dataset implementation which is used by data science computing algorithms
- This reusable design approach can be used for bigger datasets distributed over several data nodes over network.

## VII. References

[1] Buschmann, F., Henney, K. and Schmidt, D.C., Pattern-Oriented Software Architecture Volume 5: On Patterns and Pattern Languages. Wiley Series in Software Design Patterns. John Wiley and Sons Ltd, 2007

[2] Electronic Notes in Theoretical Computer Science Volume 299, 25 December 2013, Pages 53-60