Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 94 (2016) 105 – 112

Procedia Computer Science

The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016)

# Partitioning Application using Graph Theory for Mobile Devices in Pervasive Computing Environments

Nevin Vunka Jungum[a]*, Nawaz Mohamudally[a] and Nimal Nissanke[b]

[a]*School of Innovative Technologies and Engineering, University of Technology Mauritius, La Tour Koenig, Mauritius*
[b]*School of Computing, Information Systems and Mathematics, London South Bank University, London, UK*

**Abstract**

The very first phase in software partitioning is to choose an appropriate clustering methodology before moving the clusters to multiple computational nodes since this stage can impact on the distribution of clusters and eventually on the performance of the overall application. In pervasive computing environments, certain memory and cpu intensive applications would prefer to use the computational nodes (anything with at least networking, storage and cpu capabilities) available in the environment. To this end, this paper investigates the very specific area of clustering or partitioning an object-oriented application running on mobile devices. Graph theory has been used to model an application and a cluster analysis algorithm has been proposed. Details about the implementation are covered, followed by a laboratory application partitioning using the proposed approach. Researchers and software designers willing to investigate software partitioning can consider this practical and easy implementable approach.


Peer-review under responsibility of the Conference Program Chairs

*Keywords:* application partitioning; clustering; graph theory; pervasive computing

## 1. Introduction

The idea or concept behind pervasive computing[1] is now almost clear within the scientific community due to the large interest given by researchers. Software partitioning [2,3,9,10,11] as well is not a new topic; largely discussed at times

* Corresponding author.
*E-mail address:* nevin.vunkajungum@gmail.com


Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2016.08.018

when they were very essential for partitioning software to run on desktops and servers connected in wired networks. However, since the advent of next generation smart environments, existing partitioning falls short since they were primarily designed to work in wired networks where the percentage of connectivity loss is very low compared to mobile wireless networks. In pervasive environments, mobile devices running applications are most of the time not connected to a power source unlike in the old days of the desktops and servers.

This work deals with the specific aspect of efficient partitioning of object-oriented applications, Java in this case, running on limited resource mobile devices. Partitioning an application and moving the generated partitions to different computational nodes are two distinct operations. In this paper, the first part is investigated.

Clustering allows better scalability, easy routing and load balancing. The Java application is modeled as a connected undirected weighted graph and a clustering algorithm is applied to create clusters of classes. A Java-based application is composed of a set of classes (native and non-native) some or all linked to each other. Native classes describes the set of classes that are suitable to be executed on the source device such as components that directly accesses local I/O devices, for example in Java, components with native method to access local files, components that directly access device-specific information like system.properties contain specific information related to the host system and also components that directly handles the user interaction. Here linked refers to any relationship that would force either class to interact, for example, method calls. Method from one class/object calling another method from another class/object. Hence, using graph theory, an application can be modeled as an undirected graph $G(V, E)$ conveniently where $V$ is the set of vertices of the graph representing the set of classes (class level granularity) and $E$ is the set of edges of the graph representing the set of interaction (for example, method calls) between classes. The memory and CPU consumptions of each class are represented as attributes of the class. The idea behind using clustering is to group neighbored classes that have similar interaction frequencies /edges' values together. Since high interaction frequency classes would exchange messages very often, it is more appropriate to cluster them together and, where ever possible, host them on a single node that meet all hardware requirements. This approach has an immediate effect of reducing networking cost when compared to randomly partitioning the application in multiple groups of classes and simply loading them to random nodes. Considering the popularity of the Java programming language especially in the networking domain, the latter has been chosen for all implementations. But the proposed approach can be used without any hassle to build applications implemented in any other object oriented programming language.

The paper is structured as follows: In Section 2, an application is modeled as a graph using Graph Theory. Section 3 proposed a cluster analysis algorithm to partition the application in clusters. The application of the algorithm is explained and the approach is analyzed. Implementation details are covered in Section 4, whereas Section 5 describe the algorithm in action by partitioning an application. And finally Section 6 conclude this paper.

## 2. Modeling Application as a Graph

In this section, a formal modeling approach using graph theory is presented to model applications. It is important to model the application to permit any clustering algorithm to perform computation on the former. As mentioned above, applications whether mobile or desktop-based, are mostly composed of native and non-native classes. A connected undirected weighted graph $\boldsymbol{G(V, E)}$ is used to model an application's structure. The partitioning process of the complete graph classes aim into creating one $\boldsymbol{V^{native}}$ and $\boldsymbol{p}$ disjoint non-native partitions $\boldsymbol{V_1^{nonNative}, V_2^{nonNative}, \dots, V_k^{nonNative}}$, where $\boldsymbol{p}$ is a non-negative integer, to satisfy:

$$G(V_n) = V_n^{native} \cup V_n^{nonNative} \text{ and } G(E_n) = E_n^{native} \cup E_n^{nonNative};$$
$$\bigcup_{i \neq j}^{p} V_i^{nonNative} = V \backslash V^{native} \text{ and } V_i^{nonNative} \cap V_j^{nonNative} = \emptyset \text{ where } 1 \leq i, j = p \text{ and } i \neq j \quad (1)$$

Inspecting the anatomy of a Java application permit one to see a relationship/link among classes. For example, a sub-class extending a super-class or a method in one class calling a method in another class. Thus, as per the definition of connectivity[4] in graph theory, an undirected graph $G = (V, E)$ is *connected* if for each pair of vertices $v, w \in V$ there is a path from $v$ to $w$. As mentioned previously, the vertex set $V$ represents application classes whereby each vertex $v \in V$ is associated with a finite ordered list of $m$ elements, that is, an $m$-tuple $(r_1, r_2, \dots, r_m)$, where $m$ is a non-negative integer, which represent the different resources $r$ a vertex/class consumes. In this case, the following