2011

AVRF: A Framework to Enable Distributed Computing Using Volunteered Mobile Resources

Evan Arnold
earnold@pugetsound.edu

Building a Framework to Enable Distributed Computing on Volunteered Mobile Resources

Introduction

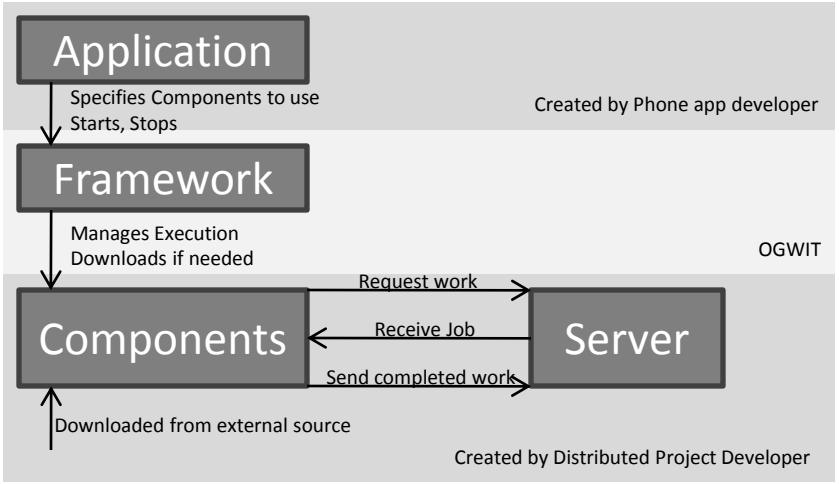
In modern computing, large scale tasks are conventionally distributed across the cores of high-powered supercomputers. An economical and environmentally friendly alternative is to distribute portions of a task across a large number of personal computers, whose owners donate the idle time on their CPU.

The recent boom in the mobile device market has created an opportunity for distributed computing. Mobile devices are continually increasing in power, yet spend the majority of their lifetime sitting idle. We created a framework that will enable developers to take advantage of some of this idle time.

Our framework provides a method to distribute computationally intense tasks to smartphones in such a manner that the increased battery drain is acceptable to the donor.

Distributed Computing

The large scale use of volunteered idle time on personal computers was pioneered by SETI@home [1]. PC owners donate their computer's processing power when not in use. SETI uses this time to analyze pieces of radio telescope readings for indications of extraterrestrial broadcasts. Users can download a screensaver which gets data to analyze from a central server, processes it, and returns the results where they are aggregated to form useful data.



Mobile Devices

Apple has sold over a hundred million iPhones, equal to around a quarter the computing power of today's fastest supercomputer, the K Computer in Kobe, Japan [2,3,4]. This does not include tablets or phones running Android or other operating systems.

Concurrent with this increase in power has come an increase in ease of development. Mobile devices today have sophisticated APIs and allow developers to broaden the usefulness of the devices.

The increase in power, prevalence and ease of use of mobile devices are making them a viable option for distributed computing.

Project Design

AVRF (Android Volunteered Resource Framework) is a framework that allows for the easy execution of distributed jobs in the background of UI intensive Android applications. AVRF provides the developer of an Android application with a way to easily obtain and run a set of components that work on a distributed computing project. These components recieve, process, and deliver work to a central server. The developer of the distributed project need only to create these components, which AVRF manages the execution of. A simplified example of AVRF and its interactions with the application and project components is illustrated above.

For testing AVRF, two component sets were developed: one to estimate Pi, and another which tested for Mersenne Primes. Both communicate with Java Servlets designed to work with the components via HTTP. The application specifies which component set to use, which AVRF then obtains.

Results

Doing work in the background of a UI intensive task does not effectively mask battery use. Although the screen does use a considerable amount of power, running the processor constantly entails a noticeable amount of further drain. As such, the best way to implement a distributed system using mobile devices would likely involve only doing calculations while phones are plugged in. A feasible concept, as [how much the phones can do in 8 hours or something]

Acknowledgements

Research made possible by a University of Puget Sound Science and Math Research Grant and an Amazon Web Services Cloud Computing Grant, aided by my advisor Jason Sawin, my lab mates Doug, Aryn and the Kyles, and the power of friendship.

References

[1] SETI@home: http://setiathome.berkeley.edu/
[2] Eric Strohmaier, Japan Reclaims Top Ranking on Latest TOP500 List of World's Supercomputers
[3] Christina Warren, Apple: 100 million iPhones Sold
[4] xyster.net: http://www.xyster.net/blog/?p=40

Reducing Bitmap Size Using Variable Length Compression

Fabian Corrales

Utilizing the variations in bitmap columns to allow for greater compression.

UNIVERSITY of PUGET SOUND
Est. 1888

Introduction

Modern research endeavors generate large amounts of data. One technique to manage these large repositories is to create a bitmap index. A bitmap index is a coarse binary representation of database data. Bitmap indexes will unfortunately also contain large amounts of data, and are usually compressed. Two common compressions schemes used are the Word Aligned Hybrid(WAH) [1] and the Byte-aligned Bitmap Code(BBC) [2]. Each of these schemes has its own strengths; WAH provides faster query times while BBC provides better compression. New research has shown Gray code [3] ordering to be effective in increasing compression. Our research exploits a specific aspect of Gray code ordering to create another compression scheme with better compression than both WAH and BBC, with query times comparable to WAH's; the Variable Length Compression (VLC) scheme.

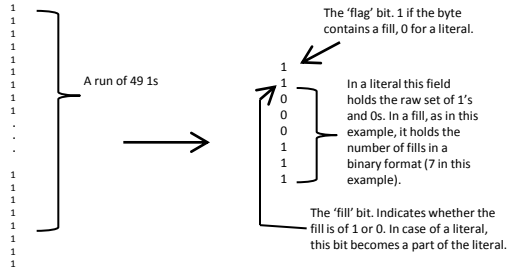


Fig. 1 An example of BBC type compression. This example shows a run with a fill bit of 1, 7 runs long (runs are 7 bits long, so 49 consecutive 1s total).

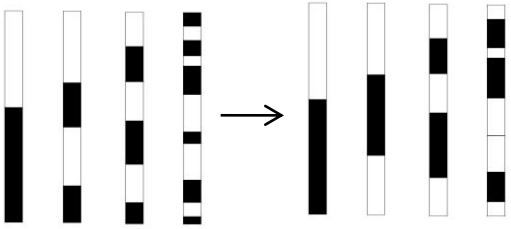


Fig. 2 Gray code ordering reorders data to allow for longer runs. The example on the left is a set of bitmaps in a lexicographical order, the example on the right is bitmaps after Gray code ordering has been applied. White blocks represent runs of 0s, and black blocks represent runs of 1s.

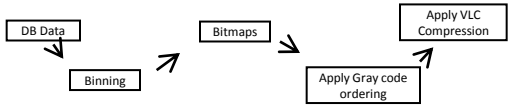


Fig. 4 Database data is first 'binned', which groups data into ranges rather than precise measurement. Bins are then transformed into bitmaps, which are reordered using Gray code, and then compressed.

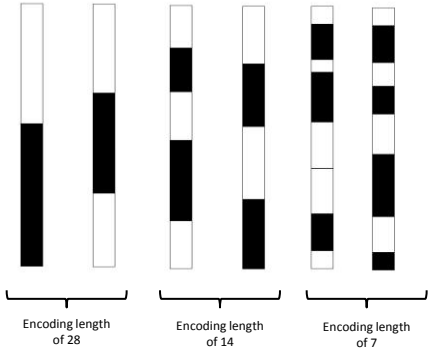


Fig. 3 The first few columns are compressed using the largest encoding length, 28. As the run lengths grow smaller, our encoding lengths will switch as necessary, to maintain efficiency.

Table 1
Compression sizes

Dataset	VLC	BBC	WAH	Uncompressed
HEP	1.12 MB	1.47 MB	2.14 MB	272 MB
STOCK	622 KB	621 KB	605 KB	6.73 MB
LANDSAT	17.8 MB	17.8 MB	26.7 MB	238 MB
HISTOBIG	573 KB	576 KB	1 MB	21 MB
UNIFORM	19 KB	30 KB	32 KB	10 MB

Table 2
Average query times in milliseconds

Dataset	VLC	BBC	WAH
HEP	194	232	226
STOCK	32	29	18
LANDSAT	436	415	473
HISTOBIG	140	145	137
UNIFORM	11	15	10

BBC and WAH Compression

WAH and BBC are lossless run length compression schemes. Their results can be queried without first being decompressed. WAH and BBC compress bitmaps using 32 or 8 bits, respectively (Fig 1). WAH's representation is the size of a typical word in a computer, and BBC's representation is the size of a single byte of data. BBC can achieve better compression than WAH, as it compresses using smaller run lengths. WAH can achieve better query times by storing runs in a word and avoiding any information parsing.

Gray Code

Gray code ordering reorders data so that successive values differ only in one bit (Fig. 2). This reordering can be applied to bitmaps because row order does not matter in a relational database. This ordering is employed as a preprocessing technique used to achieve longer runs, however, the run lengths of a Gray code ordered bitmap diminish with more columns (Fig. 2). We mitigate this effect with our new compression scheme.

VLC Compression

VLC uses three different encoding lengths: 28, 14 and 7(Fig. 3). It will determine which one of these three lengths will be optimal for a given column and then compress using that length. These lengths are used because they allow each of the larger encoding lengths to be broken down and compared with the smaller encoding lengths. This allows columns of different encoding lengths to be queried. As Gray coded run lengths deteriorate, our scheme will change encoding lengths accordingly to maintain efficiency.

Results

Tables 1 and 2 detail our findings among the data sets for compression size and query times.

Future Work

In the future we plan to modify VLC to be truly generic allowing for a greater range of encoding lengths. Other areas that can be explored are parallelization and different row ordering techniques.

Acknowledgements

This research was made possible by the University of Puget Sound McCormick research grant and aided by my advisor Jason Sawin, professor Brad Richards, my labmate John Granville, and viewers like you.

References

1) G. Antonshenkov. Byte-aligned bitmap compression. In *Data compression Conference*, 1995. Oracle Corp.
2) K. Wu, E. J. Otoo, and A. Shoshani. Compressing bitmap indexes for faster search operations. In *SSDBM*, pages 99-108, 2002.
3) A. Pinar, T. Tao and H. Ferhatosmanoglu. Compressing Bitmap Indices by Data Reorganization. In *ICDE*, pages 310-321, 2005.