Analyzing Code-Data Coupling for Optimal Partitioning and Placement

Lee Beckman – Research Proposal

Problem Statement

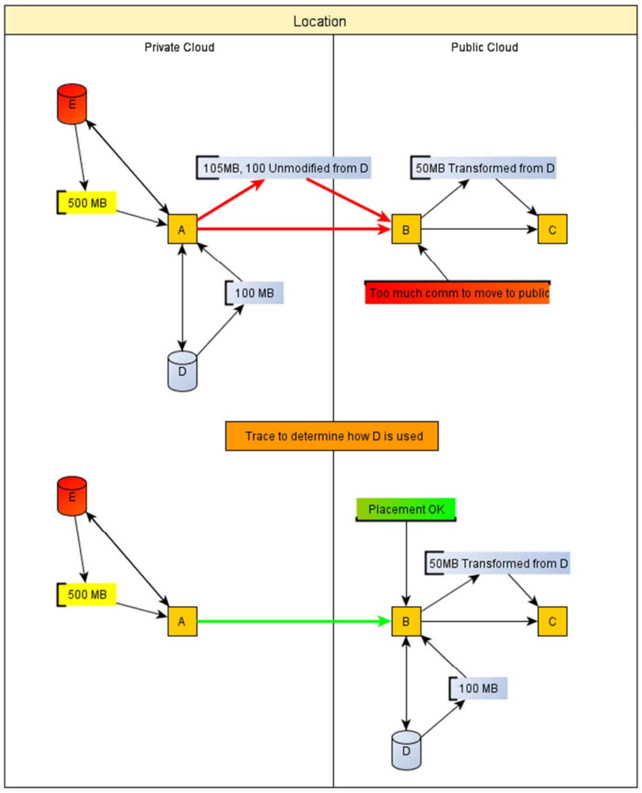

Figure 1: Data relocation in hybrid deployment by taint tracking

[CONSIDER REMOVING] Cloud computing has reached the point, through sufficient study and industry support, where it is an option that should be considered when planning and managing large-scale applications, particularly those attempting to serve an arbitrary number of users. The case has already been made in research and by example as to the potential benefits of deploying in the cloud [6], but the matter is less clear when it comes to if, when, and how existing applications can be migrated to the cloud. This is largely due to the fact that existing applications often need to be re-architected to better exploit the kinds of resources available in the cloud or to meet certain constraints placed on components of the application – such as in a hybrid deployment where only certain parts of an application are moved to the cloud. If IT managers and system architects are given the proper tool support to help automatically determine how an application may be beneficially moved to the cloud while satisfying business constraints, more applications will realize these benefits. Much of the research to support this deals with automatic application partitioning [1, 3, 4, b-4, 11, 12, 13, 18, 20], which generally seeks to divide a more monolithic application into components which can be separated; and placement [8, 10, 14, 15, 16, 17, 19], which decides on the best places to put those components – be they private server installations or various public clouds with heterogeneous resource offerings. An aspect of these processes which has have not been sufficiently addressed in the literature is how to account for an application's persistent state. The most obvious problem is dealing with the potentially massive volumes of relational data that are a part of many applications. Existing research has primarily focused on how the components of an application consume CPU and memory and how much components communicate with each other under varied partitionings. In addition to component-component coupling, we must consider component-data coupling. If applications are to be partitioned, and these partitions relocated, the interactions that the application had with large, difficult-to-move bodies of data will be affected and are critical to understand. In particular, it would be useful to analyze how an application's components interact with persistent state under various workloads. This would allow for more intelligent decisions to be made with respect to component placement, data placement/movement, and even data partitioning. Database partitioning is nothing new [b-1], but a partitioning strategy which has at its core a consideration for the placement of the components which depend on the data is worth investigating. As a basic motivating example, consider Figure 1, which depicts a hybrid deployment between a private and public cloud. In the upper portion, taint tracking has determined that A must remain in the private cloud due to a heavy dependence on data from E, which is pinned to private. More importantly, the tracking shows that A is basically acting as a relay in passing data from D to B, and that this data represents a significant communication overhead between A and B. Thus, this deployment which may be beneficial due to B and C being offloaded to the cloud is deemed undesirable. However, since tracking has determined that A merely acts as a relay, a refactoring is suggested which allows B to directly access D itself. This effectively gives more freedom to the placement of D, and makes the deployment of A, B, and C possible.

[CONSIDER REMOVING] It should be noted that this problem is stated as a starting point for the research. Investigating this could lead to deeper considerations of persistent state under partitioning, such as how to treat persistent state within memory, and how to analyze state in general to answer questions about the replicability of components.

To summarize the problem statement:

The value of partitioning applications within and between clouds has been demonstrated, and now there is a need for tools which can automate the discovery and placement of application components to optimize performance and cost. Due to expectations to scale to heavy user demand and the high cost of communication, persistent data is a frequent bottleneck and something which must be considered carefully to achieve the best partitionings. There is a need to understand how modules interact with persistent data, modify it, and share it with other modules. Following that, there is a need to use this knowledge to attempt to give more freedom to the placement of data and modules in a cloud deployment.

Related Work

This research proposal is influenced by work in the areas of application partitioning, database as a service, and dynamic taint tracking. The following gives a brief overview of some important works which serve to establish the potential novelty and value of the proposed work.

[REVISE] In [6], Hajjat et al. demonstrate the benefits of a hybrid cloud deployment over an "all or nothing" migration, showing that cost saving are possible for enterprises without losing performance beyond acceptable levels. They touch slightly on the importance of databases, noting that they may contain sensitive data restricting their placement and suggesting that components interacting extensively with the database should be placed with them to save on communication costs. However, since they partition at the level of entire servers they only allow migration of entire databases, which is not as flexible as we'd like to consider. Partial migration of data is mentioned as an interesting future consideration, and they also leave handling dynamic workload variations as an open issue.

Database as a cloud service is explored in [2], which presents a relational system in the cloud capable of automatic partitioning at the granularity of tuples. They argue that partitioning and load balancing of databases is necessary as systems scale, but that doing it manually is very difficult for even experienced users. They then show that their prototype has the potential to offer automatic scaling performance. As in [6], the importance of online workload analysis is discussed as they note that under different loads an application may have different requirements on data location. This work does not consider the partitioning of application code and how such could relate to data placement, but does demonstrate that automatic analysis of how data is used in the cloud is important.

At a higher level, in [9] Khajeh-Hosseini et al. attempt to identify open problems for enterprise cloud computing research. Moving data around in the cloud is given as a major bottleneck, and they suggest that enterprises may wish to indicate where data should be made available and how it can be moved around. They importantly note that tools need to be developed to help in migration decisions – tools which provide models of data as one of their components.

Greenberg et al., in [4], highlight "geo-distributing state" as an open problem for clouds, and argue that everyone is more or less coming up with their own solutions. They see this as an opportunity to develop general-purpose tools, mechanisms, and APIs. Looking to the strategies other companies have used they note that there are various possible ways to manage data with tradeoffs depending on how it is typically accessed, which supports the development of analysis tools to help choose the optimal strategy.

[CONSIDER REMOVING] Even outside of cloud computing, the automatic identification of persistent state is an issue. The Galapagos tool [7], inspired by experience in application migration, is used to understand how large enterprise systems use storage. The authors focus on the need to automate these tasks due to their complexity and the fact that such systems are often poorly documented and understood. In one analysis their tool discovered mismatches between the importance of business data and the quality of the storage upon which it was served. They believe that such automated analysis will be important for enterprise optimization for at least a decade.

These preceding works serve to make the point that the treatment of persistent data is an important issue, and especially so in the cloud. Furthermore, it is a problem of sufficient complexity to demand tools which can perform useful analyses automatically. What must now be shown is that in existing research on migrating applications to the cloud, there is room to improve the handling of persistent data.

To start, let us consider various recent systems designed to perform application partitioning, a key enabler of such cloud computing models as offloading for resource-constrained devices and hybrid deployments for large enterprise applications. AlfredO [13], Hilda [20], SWIFT [1], J-Orchestra [18], and [3, 5, 11, 12] all attempt to enable applications which execute across multiple hosts. There are many restrictions amongst these systems, such as requiring the use of a specific language [1, 11, 20] or being focused on offloading for mobile devices, but the main limitation is that none of them consider the placement of data when partitioning applications. The focus is rather on the exchange of data between code modules themselves, as well as on the CPU and memory usage of components. These are necessary to consider, of course, but one must go further to provide a complete picture of the implications of partitioning an application.

Systems which focus on resource provisioning, which deal with both carefully reserving resources and then efficiently using them, are perhaps more likely to address the persistent data. Provisioning is related to partitioning as a kind of complementary next step, taking a partitioning and assigning its components optimally to a set of hosts. As the focus is on resource usage, it may be that disk interaction receives more consideration.

Looking to some recent works, however, in particular [8, 17, 19], the consideration of data placement is absent amongst the analyses of CPU, memory, and inter-component communication bandwidth.

In [16], Stewart and Shen do consider persistent data on disk when placing components; however the data they collect to evaluate costs is only at the level of disk bandwidth. They do not determine what data is actually used by which components, but simply how heavily a component uses physical disks. Furthermore, their analysis is offline, which misses the opportunity to track an application as it runs to adjust to changing workloads on the fly, as well as learn the nature of typical workloads themselves. Shimizu et al. [15] are limited similarly by considering the cost of disk access only in terms of bandwidth, treating it much like a simple CPU usage measurement. Again, in [14], the CAFe system attempts to optimally place database tiers, but does consider the contents of such databases, and thus no decisions about partitioning and placing tables near the components which actually use them are possible

The system by Li et al. [10] perhaps comes closest to our aims. Given a Layered Queuing Network model of an application, it attempts to find an optimal placement of application components. The LQN model can express disks and even such entities as relations in a database, and so fine-grained positioning of persistent disk state can be addressed. However, this approach is limited by the LQN itself, which is not generated automatically (requiring deep understanding of an application to create) and which cannot change dynamically. This means that the analysis is too rigid, and possible partitionings of a database cannot be discovered and analyzed automatically.

Added sources:

In [b-4], the Coign system is presented. The paper is notable in that it deals with relating application executions in terms of stack trace-like hierarchies to the requests which generate them. This can be useful both for obtaining an overview of how applications interact with data to drive a deeper analysis, and also for potentially directing requests to deployments optimized to serve a particular request form. Furthermore, this paper makes an interesting observation when partitioning an application across tiers that, contrary to programmer decisions, certain data-caching components should be moved to other tiers to be closer to the components which use the data. This is similar to the idea covered in Figure 1, a potential scenario that I seek to address.

Wiedermann et al., in [b-5], perform a static analysis to transform programs which operate on relational data to execute explicit queries. As an example, consider a simple program which queries for a set of results, and then programmatically filters those results in a loop based on some predicate; their system would transform this program to instead do the filtering as a part of the initial query, if possible. This work is interesting as it provides an example of an advanced method to consider how persistent data is used and modifed by an application, even across procedural boundaries, and how it affects an application's execution.

Considering dynamic taint tracking, it is a technique which has been almost exclusively used in a security context to analyze the flow of untrusted/secure data [b-2, b-3, b-6]. To my knowledge, it has not been used in the way I am proposing, to analyze use of data and potential communication overheads.

Approach

The approach will be directed by the following thesis:

Better hybrid cloud deployments can be realized by giving more freedom to the placement and use of persistent data in partitioning schemes. This can be addressed by using dynamic taint tracking to analyze how an application's modules interact with persistent data and suggesting modifications to these interactions.

To start, I will find a dynamic taint tracking system for Java programs. An off-the-shelf solution is greatly preferable, with little to no modification by myself, as taint tracking itself is no longer an interesting problem.

The analysis enabled by such a system will eventually be used to capture two key elements of application execution:

i) The nature of the requests being made to an application, assuming a request-response, web-application type architecture

ii) How relational-database data is used for a particular request. Namely, will want to know:
   a. How much/when data is read/written from a particular table by a particular class
   b. How much of this data is passed to other modules, to determine the contribution of it to potential communication overheads
   c. How this data is modified by various classes, including concatenation, being wrapped in objects, filtering, and transformations, to determine points where data dependencies can be potentially modified
   d. How this data affects the execution flow of the program, again to potentially modify data dependencies

Even before beginning to collect this kind of data, however, a more simple analysis will be used. Drawing on techniques similar to those presented in the Coign paper [b-4], where stack-trace-like summaries are created and grouped for various executions of an application, and using manual code inspection, I will analyze a series of applications. The goal will be to quickly gain insight into how various applications actually interact with databases to see if the more detailed analyses described above could actually be useful. As a starting example, I would attempt to find cases where the scenario shown in Figure 1 applies. AspectJ could potentially be used for this analysis, and other pre-existing tools will be considered as necessary. For applications to analyze, RUBiS – being well-studied and easily available via Nima – will be a good start, followed by applications which will need to be decided on and collected.

Once the value of an in-depth analysis of persistent data usage is established and analysis data is being collected, I can explore generating useful suggestions on application modification based on the data. These could include, but are not limited to:

i) Refactoring modules to move dependencies on data sources from one module to another. One scenario for this is when a module is simply acting as a relay for data, incurring unnecessary communication overhead.

ii) Integrating with Nima's partitioning and cost analysis framework to determine how data placement affects performance and cost.

iii) Discovering possible database partitionings.

iv) Attempting to identify opportunities for replication/caching based on how data is used.

This approach represents a starting point, and depending on how useful the analyses turn out to be it may be subject to change and extension. One interesting idea which has not been discussed yet are systems to track how an application uses persistent data stored in memory, such as in session objects, and the implications that this might have for a cloud partitioning and placement system.

Works Cited

[1] Chong, S., Liu, J., Myers, A. C., Qi, X., Vikram, K., Zheng, L., et al. (2009). Building Secure Web Applications with Automatic Partitioning. *Communications of the ACM*, 79-87.

[2] Curino, C., Jones, E., Zhang, Y., Wu, E., & Madden, S. (2010). *Relational Cloud: The Case for a Database Service.* Cambridge: MIT.

[3] Diaconescu, R., Wang, L., Mouri, Z., & Chu, M. (2005). A Compiler and Runtime Infrastructure for Automatic Program Distribution. *19th International Parallel & Distributed Processing Symposium.* Denver: IEEE Computer Society.

[4] Greenberg, A., Hamilton, J., Maltz, D. A., & Patel, P. (2009). The Cost of Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 68-73.

[5] Gu, X., Messer, A., Greenberg, I., Milojicic, D., & Nahrstedt, K. (2004). Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 66-73.

[6] Hajjat, M., Sun, X., Sung, Y.-W., Maltz, D., Rao, S., Sripanidkulchai, K., et al. (2010). Cloudward Bound: planning for beneficial migration of enterprise applications to the cloud. *SIGCOMM*, 243-254.

[7] Joukov, N., Pfitzmann, B., Ramasamy, H. V., & Devarakonda, M. V. (2010). Application-storage discovery. *SYSTOR*.

[8] Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., et al. (2006). Dynamic placement for clustered web applications. *Proceedings of the 15th international conference on World Wide Web* (p. 604). ACM.

[9] Khajeh-Hosseini, A., Sommerville, I., & Sriram, I. (2010). Research Challenges for Enterprise Cloud Computing. *CoRR*.

[10] Li, J. (., Chinneck, J. W., Woodside, C. M., & Litoiu, M. (2009). Fast scalable optimization to configure service systems having cost and quality of service constraints. *6th international conference on Autonomic computing* (pp. 159-168). ACM.

[11] Nanda, M. G., Chandra, S., & Sarkar, V. (2004). Decentralizing execution of composite web services. In *OOPSLA* (pp. 170-187). ACM.

[12] Ou, S., Yang, K., & Zhang, J. (2007). An effective offloading middleware for pervasive. *Pervasivev and Mobile Computing*, 362-385.

[13] Rellermeyer, J. S., Riva, O., & Alonso, G. (2008). AlfredO: An Architecture for Flexible Interaction with Electronic Devices. *9th ACM/IFIP/USENIX International Conference on Middleware* (pp. 22-41). New York: Springer-Verlag.

[14] Roy, N., Xue, Y., Gokhale, A. S., Dowdy, L. W., & Schmidt, D. C. (2009). A Component Assignment Framework for Improved Capacity and Assured Performance in Web Portals. *Proceedings of the*

Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on the Move to Meaningful Internet Systems (pp. 671-689). Berlin: Springer-Verlag.

[15] Shimizu, S., Rangaswami, R., Duran-Limon, H. A., & Corona-Perez, M. (2009). Platform-independent modeling and prediction of application resource usage characteristics. *Journal of Systems and Software*, 2117-2127.

[16] Stewart, C., & Shen, K. (2005). Performance Modeling and System Management for Multi-component Online Services. *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (pp. 71–84). Berkeley: USENIX.

[17] Tang, C., Steinder, M., Spreitzer, M., & Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. *Proceedings of the 16th international conference on World Wide Web* (pp. 331-340). New York: ACM.

[18] Tilevich, E., & Smaragdakis, Y. (2002). J-Orchestra: Automatic Java Application Partitioning. In *ECOOP* (pp. 178-204). Springer-Verlag.

[19] Urgaonkar, B., Rosenberg, A. L., & Shenoy, P. J. (2007). Application Placement on a Cluster of Servers. *International Journal of Foundations of Computer Science*, 1023-1041.

[20] Yang, F., Gupta, N., Gerner, N., Qi, X., Demers, A., Gehrke, J., et al. (2007). A Unified Platform for Data Driven Web Applications with Automatic Client-Server Partitioning. *16th international conference on World Wide Web* (pp. 341-350). ACM Press.

To Be Added

[b-1] Bowman, I. T. (2001). *Hybrid Shipping Architectures: A Survey.*

[b-2] Chang, W., Streiff, B., & Lin, C. (2008). Efficient and extensible security enforcement using dynamic data flow analysis. *ACM Conference on Computer and Communications Security* (pp. 39-50). ACM.

[b-3] Chin, E., & Wagner, D. (2009). Efficient character-level taint tracking for Java. *SWS* (pp. 3-12). ACM.

[b-4] Hunt, G. C., & Scott, M. L. (1999). The Coign Automatic Distributed Partitioning System. *OSDI*, 187-200.

[b-5] Wiedermann, B., Ibrahim, A., & Cook, W. R. (2008). Interprocedural query extraction for transparent persistence. *OOPSLA*, 19-36.

[b-6] Zhu, D. (., Jung, J., Song, D., Kohno, T., & Wetherall, D. (2011). TaintEraser: protecting sensitive data leaks using application-level taint tracking. *Operating Systems Review*, 142-154.