OODBMS Architectures Defended

By Robert Greene, Versant Corporation

It is the intent of this paper to address the issues raised in "OODBMS Architectures Revisited".

As a preface, the intent of the original article "OODBMS Architectures", was to offer an adopter of OODB technology practical knowledge that would facilitate a successful adoption. Practical not theoretical advice, because at the end of the day, each of us must create scalable systems around the practical limitations of the individual components of those systems. And so, I think it is important to comment on Mariott's rebuttal which contains, in my option, a lot of theoretical content which is less than useful for practical implementations. In addition, I want to clarify comments Mariott made which are incorrect or out of context, while recognizing areas where his comments raise valid points.

Along the way, I will attempt to keep this short and to the point as I've already expressed my views on the subject in the original paper. I will not attempt here to go over the specific capabilities of the Versant OODB as that information is adequately covered in Versant's user manuals.

Response to "First Paragraph, Page 3":
Here is appears Mariott is challenging my assertion that OODB architectures have tremendous impact on application scalability. Clearly the context of application characteristics, found in the sentence above the one examined in "revisited", was ignored in the authors review comments while extracting my sentence from the containing paragraph. In the original paper, application characteristics, which ultimately drive architectural decisions were equally emphasized, though frankly I would assume this to be obvious to my readers. I stand by my statement, OODB's exhibit very different characteristics, an adopter should consider the application characteristics and match those with the OODB architecture best suited for it's needs.

It is my opinion, the continuing discussion on proper use of OO analysis techniques should be well understood by the typical reader of these papers and is therefore irrelevant in response to my original writing.

Response to "Page Based - Last Paragraph, Page 4":
It looks like the issue here for Mariott is the part of the sentence using the words "*have* to be implemented" regarding ObjectStore's object placement strategies. Again, I sought to provide practical advice, not theoretical. I will stand corrected and qualify, it is my opinion, only the most trivial demos using a page based architecture could ignore the object placement issue. In this paper, I will show where Mariott points again and again to this object placement strategy as the means to avoid various design pitfalls. So, practically speaking, it's a must have in a real world application where you will have significant concurrent users and large databases and Mariott is validating this point by his very writing. If an adopter should choose to ignore this issue it will certainly be to their peril

Response to "Page Based - First Paragraph, Page 5":
Regarding the issue of query execution locality as pertains to your application requirements. I think it was obvious to my readers that the application characteristics play a significant part in the role of query locality. As Mariott suggests, this is no doubt a complex issue. Locating data and/or indexes at the client for query purposes may be perfectly acceptable for a certain category of application. In fact, that was the point of the original paper, make sure to consider your application characteristics and match those with the OODB with the best supporting architecture. If an application is dealing with data sets that are very large, then I would assert you do not want to be localizing to the client, though you would likely want to partition and distribute. This ties in with my point regarding one OODB architecture's ability over another to handle larger data sets. Theoretically, I could drag 5 gigabytes of indexes into all my client processes and manage their update across the distribution, but as a practical matter, a developer could lose his/her job for trying to implement that solution.

Response to "Concurrency Model - Paragraph 2, Page 5":

Apparently, Objectivity and Versant have the capability to relax ACID properties, but ObjectStore does not have this ability. So, I stand corrected. What was Mariott's point here anyway, when in the ensuing sentence I continued by stating that my discussions were centered around proper ACID behavior. The quoted sentence "*updates will always cause lock coordination and cache consistency operations"* was taken out of context by Mariott.

Clearly, what was meant here, in context, was that a TRANSACTION which will exhibit ACID properties and is doing UPDATES will require lock coordination and cache consistency operations. If we are talking about individual operations within a transaction, then there are lots of possibilities. To that end, I'm not sure what was Mariott's point about millions of objects needing to be locked and the impact of that in the object based architecture. In a practical implementation, the GateKeeper locking pattern would be used in these cases. For example, if the only access to the thousands of atoms that make up a molecular compound are through the compound, then I only need to lock the compound, not the atoms. I recall in the early days of relational databases where various vendors made similar arguments about lock granularity, and now a decade later all dominant vendors implement record level locking. I would not suggest that this topic is complete from either my example nor Mariott's. However, contrary to the authors assertion, I did not *"omit"* information regarding application locking semantics. Even an optimistic locking transaction requires transitional locks to get a logically consistent snap shot. Exploring all the possibilities here would result in a book, of which a number have already been written.

In a further quote from the author, *"Greene also fails to observe that in page based systems where you really need object level locking you can simply place an object on a page by itself."* Well then I guess I also forgot to talk about wasted space within pages when storing a 20byte object on a 4,096 byte page, memory consumption implications in the address translation process, wasted bandwidth moving essentially hollow pages across the network ….. all so you can have a granular lock in a page based architecture. I will let the reader decide here what is practical versus theoretical.

Response to "Page-Concurrency - Paragraph 2, Page 6":
With regard to locking semantics and implications for highly concurrent applications. A quote from Mariott, *"Greene maintains that this presents a potential problem. How serious is this in reality?"* Well now I guess that really depends on whether or not you've used that -not required- object placement strategy and of course, even the author points this out! While the rest of this section is a great explanation of how ObjectStore's locking might get impacted by CRUD operations and how their -not required- clustering and object placement strategies can be used to avoid the negative impact, when Mariott offers that *"In real-world systems, most database access typically consists of 'mostly read' scenarios",* he must be referring to his experience.

Response to "Concurrency Model Implications - First Paragraph, Page 7":
It appears to me, Mariott is assuming he's used a great -not required- object placement strategy. When the author states, *"Greene also fails to note here that the same thrashing can occur at the object level in systems which use object-level locking",* he is incorrect. I did not state this, because if the object based architecture has this conflict, it is handled in a lock request queue in the server and it is not required to communicate callbacks to all clients who have that object cached as is required in the page server. I again was offering practical advice illustrating the importance of placement strategies in the page based architecture which is why I followed with the unquoted *"Applications which have relatively fixed models, well segmented into an independent cluster, with mostly non-conflicting updates, will generally do best in page and container based architectures".*

Response to "Concurrency Model Implications - Paragraph 3, Page 7":
The author states, *"What Greene fails to note here is that server centric systems, such as Versant, can quickly exhibit scalability issues as the number of users increase because much of the processing has to occur on the server",* which is completely incorrect. That is the behavior of a relational system which only uses query to access all data and is the center point for my use of the term "balance" in the original paper. Relational databases do everything (query) in the server process, some OODB architectures, like ObjectStore's, are opposite to that approach and do everything in the client process, Versant and it's object

based architecture strikes a "balance" and does server side queries with client side navigation and caching. A Versant developer uses query only to initiate a use case, bringing in key objects related to others needed in the transaction. Then related information is navigated to on a lazy load or prefetch basis without using query processing. I will also point out that the initiating query is not "required" and that named roots can be used to immediately navigate, but experience has shown that real world applications cannot be over simplified and need both of these techniques.

I would have seriously thought Mariott, an expert in OODB's, would have realized Versant has these navigation capabilities and as such I question the motivation of his discussion on this topic. I do agree with Mariott's point that *"which is best will heavily depend on the context".* Again, that was the whole point of the original paper, understand your application characteristics and pick the right OODB architecture that matches its requirements. I simply want adopters to succeed, not to fool them into thinking a particular OODB is ALWAYS the best. I will not get into specific implementation possibilities like distributed parallel queries as these topics are well covered in Versant manuals.

Response to "Network Model Implications - Paragraph 3, Page 9":
Again, the author talks about using those -not required- object placement strategies to solve the problem.

Response to "Network Model Implications - Paragraph 2, Page 10":
Again, those -not required- placement strategies come to the rescue.

Response to "Network Model Implications - Paragraph 4, Page 10":
With regard to Mariott's assertion that the object based implementation is not the best suited for large scale highly concurrent user applications, due to server side bottlenecks and small object locking. I already explained that Versant uses navigation not query as the primary means of object access. I've also stated that there are many locking patterns found in numerous texts to address the small object issue and I won't even begin to start talking about optimistic locking transactions where timestamps instead of locking are used for consistency, there is another book somewhere on these topics. That being said, just to be clear, I did not say that queries, when defined, don't need to be well thought out, indexed, optimized, etc to provide best results. However, I am assuming in the original writing the reader knows these basics of databases access.

Response to "Page Query - Last Paragraph , Page 11":
With respect to page-based query execution behavior. I agree that this statement did not consider the indexed case and was an oversight on my part. That being said, I would rephrase with the addendum: "if indexed, all the index pages need to get loaded across the network and once the query is executed and results determined, those pages holding results which where not previously loaded across the network would now be loaded across the network on access to objects in the result set and that would result in many objects which did not really satisfy the query getting transferred across the network". And contrary to the authors assertion, I do know that ObjectStore uses indexes, as I quote the problems of index maintenance in the page based architecture as a drawback to it's design in the section titled "Query Model Implications". Based on Mariott's comments, it appears he must have missed that section.

Further, to quote the Author, *"However, contrast this with a server-centric architecture where all the queries must be run on the server machine, there is little option to distribute these queries out and exploit the genuine parallelism of multiple machines when this make sense"* Of course, I've already pointed out that Versant does not exhibit the same characteristics as relational databases in this area, but in addition it is worth pointing out that Versant is perfectly capable of treating many physical databases, running on different machines, as a single logical entity which can be queried in a parallel, so apparently there is more than a little option. I will assume that the author is simply not familiar with the specifics of the Versant OODB. I will revisit this capability in more detail later in this paper.

Response to "Object Query - Paragraph 2, Page 12":
I am not sure the relevance of Mariott's lesson on object oriented programming as a rebuttal to my original paper. Perhaps, I should have mentioned the word "optimizer" in the Page-Query section. I certainly did not mean to imply that what ever query facilities exist across the various vendors could not be optimized.

Otherwise, the rest of this section is an excellent overview on how ObjectStore query works, though I would not confuse the use of an STL collection as an excuse for client side query. Native language structures and multiple model patterns can indeed provide excellent navigational tools to avoid query and whether or not they are based on indexes and/or query semantics is yet another topic.

In no way did I suggested, as Mariott asserts, that the page based architecture could not use multiple threads. With regard to parallel distributed queries, I was not referring to taking data from a single physical database and spreading it across multiple clients for query execution, I was referring to having a single client being able to query across multiple physical databases. The issue was one of transferring needless information to clients simply to perform a query and get back a few qualified objects. So, the case that was interesting, was when I have lots of data (billions of instances), distributed across many physical databases (or even 1 for that matter), and being able to ask a simple qualified question and get a couple objects back, from which I would start navigating while performing the use case logic. What I was saying is that it would suck to have to move those billions of instances (ok, or their indexes) out to a bunch of client machines where each operates on a segment of that data and then returns the results to what (I don't believe ObjectStore provides remoting of queries across client processes) and then to actually use the results and again if not already loaded have to load the pages across the network transferring some objects that did not even satisfy the query. Of course, that last part, transferring needless objects, might have been an advantage, if I had used that -not required- object placement strategy and my ensuing use case needed them.

Response to "Query Model Implications - Paragraph 4, Page 12":
To clarify my treatment of navigational access, which Mariott claims is *"not addressed adequately".* The original paper's intent was to point out major differences between OODB's and other than the semantics of implementation, navigation is one thing they all do very well. I mean very well in the sense that the efficiency with all implementations using essentially statically defined relationships as opposed to dynamically defined relationships (calculated JOINS) in relational technology. I saw no reason to go into this topic, it is well covered in all vendor manuals.

It is my option, the author questioning the importance of the query model in "*calls into question the implications of the query model as a whole",* exhibits a serious lack of understanding about the value in such a tool for efficiently entering a use case. The query model, it's implementation, it's efficiency, make a huge difference as the data sets involved in an application get larger. The lack of appreciation for this supports my assertion that the page based architecture is the least suited to handle large data applications. Otherwise, most of this section is a nice treatment of the value of navigation in ALL OODB's.

It does appears that I was incorrect in stating that there was NO WAY to access objects not contained in query capable collections. So, I stand corrected, it seems in ObjectStore you can load all objects of a class into your client and start scanning through them until you find the ones of interest. However, the practical reality, not theoretical, is that despite the "possibilities" as described by Mariott in this section, if you try to download a page based implementation today, you will not be able to do a server side query, meaning you will have to load extraneous info to the client to proceed. Further, if you are going to do anything in a practical way from a performance perspective or memory perspective, then your objects better be in one of their query capable indexed collections, otherwise you will have to load tons of information to your client to do a sequential scan searching for your objects of interest. I will let the reader think about how this impacts basic reporting requirements.

The debate about non-model referenced objects needing automatic garbage collection, ala the behavior of a virtual machine runtime, gets religious in nature and is difficult to discuss without a dedicated paper. I think it would suffice to say that many industry luminaries go as far as to say that data should "never" be deleted. In fact, a luminary from Microsoft made this very statement in the Expert Panel on Objects And Databases at the 2007 OOPSLA Conference.

With regard to parallel execution of queries across multiple physical nodes, it is difficult for me to ascertain from Mariott's text if I am in fact wrong in my statement. If I am wrong, then I would say that it must be left up to the application developer to design this behavior rather than something ObjectStore does out of the box. In Versant, a single thread of execution, in a single transactional and cache context, issues a query to a abstraction that is a logical database. Internally, Versant uses a separate thread for each physical database in that logical representation and executes in parallel and on return does a merger so the calling thread gets one result set. It looks to me as though to do this in ObjectStore, you would have separate threads within separate caches (sessions) calling their own query and then the application would have to manage those threads and somehow the results would need to get accessed as an aggregation using ObjectStore soft pointers across the different cache contexts. I will quote the ObjectStore data sheet (http://www.progress.com/realtime/docs/datasheets/products/objectstore_datasheet.pdf), so you can see what I mean: *"A single ObjectStore session can have multiple threads sharing a single database transaction or one process with multiple sessions running multiple independent transactions. This provides ideal database support for the many threads of large-scale application servers that must service many simultaneous requests."* An ObjectStore session is a cache context.

Response to "Identity Management - Paragraph 5, Page 13":
With regard to the ability to scale just because you support 64 bit machines. It is not just the physical identity that causes the scalability issue, it is the baggage implied by physical identity and the specific implementation. I will concede, with a 64bit implementation, theoretically larger scale can be achieved with a page-based design. So, does this mean that all applications that are recompiled for 64bit will automatically scale into the terabyte range, I think not. So, perhaps this is not really a page based architecture issue, but an implementation issue. If you look at real world applications deployed today in the multi-terabyte range, using an object database as the database of record, you will find predominantly Objectivity and Versant. There is a reason for this fact, and although in theory 64 bit provides room to scale for larger data sets, I will stand by my practical statement that the page based implementation is not well suited for these larger data sets as the database of record. Versant and Objectivity were handling these data sizes long before 64 bit machines were in production.

Response to "Physical Identity - Paragraph 6, Page 13":
With regard the Mariott's assertion that I've neglected to point out the *"very fast"* dereference characteristics of physical identity. I question and so suggest the reader question, how does one qualify an implementation as *"very fast"*, when you are required to first load everything to the client. In Versant, at that point, you can pin objects in memory and just use memory pointers and not even bother with database identity. As I said, I believe all OODB's do navigation well, but I do not agree with Mariott's position here and perhaps I should have addressed this issue of navigation in more detail.

Response to "Physical Identity - Last Paragraph, Page 13":
I will let the reader of OODBMS Architectures Revisited decide for themselves how flexible schema evolution is with ObjectStore. I am not an expert on the ObjectStore implementation and Mariott seems to go to great length to show his readers that schema evolution is possible with ObjectStore. I have already stated my opinion on this topic in the original writing.

Conclusion:

I think it is important to emphasize the practical nature of my original writing and repeat that OODB architectures vary quite a bit in implementation and that has performance and scalability implications when applied to the requirements of your application. Hopefully, as a result of these additional writings from both myself and the author of OODBMS Architectures Revisited, you've learned more about how OODB's work and how better to use one to your advantage. They are powerful alternatives to traditional database technology and can serve you well.