

Eat All You Can in an All-You-Can-Eat Buffet

Ratul Mahajan Jitendra Padhye Ramya Raghavendra Brian Zill

Microsoft Research

Abstract — In contrast to the efficiency-centric designs that are more prevalent today, we advocate a view of system design based on a holistic view of available resources. This view is embodied in what we call the Buffet principle: continue using additional resources as long as the marginal benefit is more than the marginal cost of using the resources. We illustrate through several examples how this seemingly obvious principle is not adhered to by many common designs and how its application can lead to qualitatively different and better performing designs. We also discuss the broad considerations that apply to designing systems that use the Buffet principle.

1. INTRODUCTION

Alice is hungry and walks into a restaurant that is serving an all-you-can-eat buffet. She wants to eat enough food so that she can avoid hunger until the next meal. Should she eat based on expected time until the next meal, or should she eat as much as she can? In this context, the second strategy is clearly superior. With the first strategy, any misestimation in the time until the next meal (or a faster than anticipated digestive rate due to increased activity levels) would lead to hunger. The second strategy provides the best possible protection against hunger that is limited only by the space in Alice’s stomach. And it represents no additional cost over the first strategy.

Surprisingly, system design often follows the first strategy today. For instance, consider the task of adding FEC (forward error correction) bits to transmissions over a noisy channel such as wireless. In current designs, the number of added bits tends to be a function of the anticipated bit error rate [2, 4, 18], and independent of the available spectrum resources. This method protects against packet loss as long as the bit error rate is below the anticipated rate, but fails when it is higher or errors are highly bursty. This failure is unfortunate because there may be available spectrum resources in the system to protect against the errors. Worse, left unused, these resources simply go to waste.

Underlying the use of the first strategy today is the designer’s desire to make efficient use of available resources. In the example above, adding the number of bits as a function of the anticipated bit error rate is perhaps the most efficient way to use the spectrum. Adding more bits might be considered as a wasteful use of re-

sources.

But as demonstrated by the two examples above, a singular focus on efficiency can lead to poor designs. Such designs can offer poor performance even when there are spare resources in the system that can enable better performance. Instead, designs that aggressively use available resources – respectively, the stomach space and spectrum in the examples – are limited only by the amount of available resource.

Based on these observations, we put forth the Buffet principle in this paper. This principle states that we should continue using additional resources as long as the marginal benefit is more than the marginal cost of using the resources. Stated differently, efficiency should not be a driving concern in situations where more of a resource can be used at a cost that is lower than the benefit from the additional use.

We present several examples of situations where the application of the Buffet principle leads to designs that are qualitatively different and arguably perform significantly better. These examples span a range of applications across a range of resource types. They include erasure coding over lossy channels, replication for reliability, managing control traffic, and speculative execution. The diversity of these examples points to the broad applicability of the principle.

We discuss broadly the situations in which the Buffet principle can be naturally applied. These include scenarios where the opportunity cost of greedily using resources can be effectively controlled. The principle is also more readily applied when the resources in question goes to waste if not used and the greedy usage does not hurt non-Buffet users if any, for instance, when the entire resource is controlled by a Buffet design. Finally, we also discuss potential limitations of this principle. These include the fact performance is now a function of the amount of spare resources and greedy usage of one resource can strain other aspects of the system.

We are not claiming that the Buffet principle should be an overarching design criteria in all situations, only that it offers a different and potentially useful perspective to designing systems. The most attractive aspect to us is that the performance of the designs that emerge from it would be limited primarily by the amount of available resources, rather than potentially artificial limitations driven by efficiency concerns. However, its full potential can only be understood in the context of concrete, practical designs. This is an active area of research

Figure 2: Illustration of scenarios where the Buffet principle applies.

of resource available, and not using it leads to unnecessary wastage. That is, the resource in question is of a “use it or lose it” variety. This fixed amount of resource could be disk space, channel capacity, etc. While the returns from using more resources is low beyond the sweet spot, the returns nevertheless represent additional benefit that should be had when the cost is low.

- The amount of resource usage needed to hit the sweet spot is hard to determine accurately because the system is very dynamic. This occurs, for instance, when the failures or packet losses are bursty; here, even if the view of average failure or loss rate is accurate, bustiness implies that at any given instance the system may be operating far from the sweet spot. In such scenarios, the system would perform better if the focus was on using as much resource as possible, to improve performance, rather than trying to operate at constantly a moving target.

We argue in this paper that instead of blindly focusing on efficiency, the designers must take a holistic look at the resources at their disposal. This can lead to designs whose performance is limited only the amount of available resource, rather than a designer-imposed constraint on the amount of resource usage. Towards this end, we outline the buffet principle in the next section.

3. THE BUFFET PRINCIPLE

The Buffet principle is easily stated: *continue using additional resources as long as the marginal benefit is more than the marginal cost of using the resources.* This principle is illustrated in Figure 2. This graphic is simplistic in that it does not capture the dynamic nature of resource availability, cost, or benefit, but it does capture the essence of the principle.

The simplicity of the Buffet principle is deceptive, to the extent that it might even seem obvious and in wide usage. But system design today is seldom approached from the perspective advocated by it. This aspect will

become clear from the case studies outlined in the next section.

For a quick illustration, however, consider TCP, the dominant transport protocol today. At first glance, it may appear that TCP embodies the Buffet principle because it tries to estimate and consume all available bandwidth. However, TCP consumes all available bandwidth only if there is sufficient amount of new data, for instance, during a large file transfer. It will not use the excess bandwidth, even if available, to protect existing data from loss. For example, consider the case where TCP's congestion window is 8 packets; assume for the sake of argument that this window is reflective of the available bandwidth along the path. Now, if TCP receives only 4 packets from the application, it will initially send only 4 packets even though the path can support more packets. If any of those packets are determined to be lost, which takes at least a round trip time, it will then send more. A transport protocol based on the Buffet principle might pre-emptively send each application packet twice, using all available resources and thus providing faster recovery against loss. Of course, whether such a transport protocol is appropriate depends on the nature of the paths and whether the additional bandwidth can be otherwise used by other flows.

Interestingly, however, one domain that does appear to embody the Buffet principle is the sale of airline seats. For each flight, airline companies have a fixed number of seats to sell. One business model is to set a fixed price at the sweet spot that balances return per seat and the number of seats sold. A different strategy, and one which the airline companies tend to use, is to drop the price closer to flying date. This is done in an effort to sell all remaining seats. As long as the price is above the marginal cost of carrying additional passengers, this strategy would be more profitable, even though its profit per seat sold is lower.

The change in perspective from a design driven by efficient use of resources to a design driven by consuming resources until the marginal benefit minus cost is positive is subtle. But as the examples in the following section illustrate, it can lead to novel designs that perform significantly better.

4. CASE STUDIES

In this section, we outline several examples where efficiency-focused designs are outperformed by Buffet designs. We classify our examples based on the nature of the resource in question. These examples are not complete designs but are meant to highlight the diversity of settings in which the Buffet principle can bring benefit. Many challenges need to be addressed before they become practical. A key challenge common to all examples is ensuring that aggressive resource usage does

not become a distraction by itself. We suggest how this challenge may be addressed in each case. We return in the next section to a more general discussion on the key considerations surrounding the application of the Buffet principle.

4.1 Wireless spectrum or bandwidth

4.1.1 Forward error correction (FEC)

Wireless medium tends to be error-prone and the bits inferred by the receiver may not be the same as those transmitted. Adding FEC bits can help recover from some of the bit errors and thus improve performance by reducing packet loss.

Conventional methods add a fixed number of FEC bits to transmissions, which is based on the expected bit error rate case. While this works for the common case, performance suffers whenever conditions are worse than the expected common case. Some recent designs tune the number of FEC bits based on estimated bit error rate [2, 4, 18]. Their effectiveness depends on the quality of their estimate. If they overestimate bit error rate and add more than required redundancy, throughput will suffer in times of high load. If they underestimate bit error rate, throughput will suffer, which is avoidable when the medium is underutilized.

An FEC design based on the Buffet principle can enable the maximal protection against bit errors that the amount of available spectrum resources can provide. Such a design, for instance, will add some fixed number of FEC bits to all transmissions, perhaps based on the expected common case. On top of that, it will add more FEC bits as long as there is available spectrum.

A key challenge here is to ensure that additional FEC bits do not interfere with other data transmissions; otherwise, overall system throughput will suffer, especially when the data traffic is high. Embedding FEC bits within data packets does not have this property. But we can accomplish the desired behavior by transmitting additional FEC bits with a lower priority such that data traffic always takes precedence over FEC traffic. Such priority mechanisms can be implemented using today's hardware that supports quality of service (QoS) enhancements (802.11e) [1].

We are currently designing such an FEC system. Our focus is VoIP and multimedia streaming applications. Aggressive addition of FEC bits in these domains leads to lower packet loss and more timely delivery of data. At the same time, lowering the priority of additional FEC bits ensures that we do not hurt other data traffic.

4.1.2 Erasure coding for lossy paths

Rationale similar to the one above also applies to protection against losses in any lossy medium, such as dirty fiber. A recent mechanism adds a fixed amount of re-

dundancy to protect against these losses [5], which renders it ineffective if the loss rates are higher than those protected against. A method based on the Buffet principle will provide greater protection during periods when there is available leftover capacity.

The key challenge here is to send redundant information such that it does not interfere with normal data traffic. One way to accomplish this behavior is to opportunistically send erasure coded data only when the queues are empty.

We are currently building such a system. Our system is targeted for paths provided by cellular providers in vehicular environments; such paths tend to be highly lossy and have unpredictable loss rates [12]. But their roughly stable capacity lets us estimate when the queues are empty and more erasure coded packets can be sent. Our coding methodology can be easily adapted for other kinds of paths such as those based on dirty fiber.

4.1.3 Mobility updates

The performance of systems that exhibit a high-degree of mobility, such as a mobile ad hoc network (MANET), depends on the frequency of mobility updates. A higher frequency yields better performance as nodes will have a more up-to-date view of the topology, but it can also swamp data traffic. Existing systems get around this trade-off by setting the frequency of updates to a tolerable level that is based on an analysis similar to the sweet spot reasoning presented in the previous section [8, 3]. Such systems may perform poorly in situations with higher than anticipated mobility levels even when there is spare capacity to support a high update frequency.

A mobility update mechanism based on the Buffer principle will provide better performance whenever spare capacity is available. The practical difficulty here again is ensuring that the additional updates do not hurt data traffic. This can be accomplished using a priority mechanism similar to the one suggested above to FEC transmissions.

4.1.4 DTN Routing

As further proof of its value, at least one recent design [6] appears to embody the Buffet principle.¹ Many DTN routing protocols replicate messages along multiple paths to improve their delivery probability. However, most protocols attempt to limit the amount of replication to prevent messages from flooding the entire network [13, 10, 14]. Because this limit is not guided by the amount of available storage at or bandwidth between replication end points, these designs can perform poorly even when plenty of resources are available. On the

¹In contrast to this work (or other specific cases that we may be unaware of), our contribution lies in an explicit and general specification of the principle and a broad discussion of its applicability, strengths and limitations.

other hand, the design of RAPID [6] uses the Buffet principle. It replicates as much as available resources allow. Messages are prioritized to prevent a handful of messages from dominating the network and crowding out others. The authors demonstrate that RAPID significantly outperforms more traditional designs.

4.2 Disk or long-term storage

Replication of data protects it against node failures or latent sector errors in disks. The amount of replication, however, is pre-determined in systems today [?]. This unnecessarily limits the amount of protection provided by replication even when there are spare resources. A replication system based on the Buffet principle will provide maximal protection given available resources.

Consider two scenarios in this domain. The first is replication across one or more disks on a single computer. Replication mechanisms today (e.g., various RAID configurations) are based on a preset amount of replication that provides protection against a certain number of failures. This can lead to data loss when more failures occur even though ample working storage may still be available. A replication design based on the Buffet principle would replicate aggressively to fill up all available storage. This design will provide maximal protection that available storage can provide. The key challenge is to not hurt read and write performance in the process of replication, which we believe can be accomplished by relegating the task of additional replication to the background and conducting it only when the disk is idle.

The second scenario is replication across computers as in a data center or a peer-to-peer system. Here too, the system will be more reliable if we were to expand the level to replication to using all available resources rather than employing a fixed amount of replication. A key challenge is to manage the impact of aggressive replication on normal data traffic because of its bandwidth needs. This may not be much of a concern inside a data center where ample spare bandwidth may be available. In situation where that may not be true, such as the wide-area context, one can reduce the impact of aggressive replication through background transfer protocols like TCP Nice [16].

4.3 Caches or short-term storage

4.3.1 Prefetching libraries and executables

The speed of program execution can sometimes be slowed by the time it takes to load in memory the program itself and the libraries it links to. One can speed this up by preemptively loading in memory commonly used binaries when there is space (and time) available. Indeed, this strategy has already been proposed or implemented for modern operating systems today [7, 15]. A strategy based on the Buffet principle will maximize

performance by aggressively fill all available memory, instead of stopping after loading the most likely candidates.

4.3.2 *Pre-fetching web pages*

Similar ideas have been explored in the context of pre-fetching web pages that users are likely to view in the future [9, 17, 11]. If the bandwidth impact of such prefetching can be minimized, for instance using a background transfer mechanism [16], such systems should aggressively fill available cache capacity to maximize user-perceived performance.

4.4 Computational resources

Speculative execution of code is a commonly used technique in modern processors [?]. In it, parts of code are executed even though the results may eventually be discarded, depending on the outcome of (if) conditions that occur prior to these parts. (The execution of the program is non-sequential to parallelize processing.) When the results prove useful, speculative execution boosts performance. The performance benefit of speculative execution depends on the accuracy branch prediction (without definitive evaluation). Conventionally, code along only one branch is speculatively executed even though additional resources may be available for executing multiple branches. For maximal performance, a Buffet design would speculatively follow as many paths as current resources levels allow. As the number of cores inside processors increase, such a design would increasingly outperform strategies that limit speculative execution to more likely paths.

5. APPLICABILITY CONSIDERATIONS

In this section, we discuss considerations related to applying the Buffet principle in system design. These are based on our early experiences with designing systems based on this principle. They will be refined as we gain more experience.

5.1 When is it safe to greedily use resources?

The glib answer to this question is embedded within the principle statement itself: whenever the marginal cost is more than the marginal benefit. On a deeper level, however, quantification of the benefit and cost can be tricky. The primary difficulty is accounting for the opportunity cost of greedily using resources, that is, for what else could those resources might have been used. This is not a concern where the greedily allocated resource can be easily reclaimed or the resource would otherwise remain unused, as is the case for a resource like disk blocks. But this is a major concern otherwise. For instance, greedily transmitting more FEC bits uses resources that might have been otherwise used for trans-

mitting more data, which can hurt overall system performance. The amount of FEC that can be safely added is thus dependent on the amount of data that applications generate. But predicting the amount of application data can be error prone.

In our designs, one effective way that we have found to skirt this difficulty is to greedily use resources in an opportunistic manner that minimally interferes with the normal operation of the system. For our FEC design, we send additional FEC bits only when the spectrum is vacant and with a low priority; this ensures that new data gets higher priority over the greedy FEC bits. For our erasure coded design, we send additional erasure coded packets only when the queues are empty, such that new data is never delayed by coded packets. We believe that such opportunistic usage can pave the way for greedy use of resources in other settings as well.

5.2 For what types of resources is the Buffet principle most useful?

Two categories of resources are ideally suited for the application of the Buffet principle. First, the principle appears more useful for non-conservable resources, or those that would go to waste if they are not used. Storage, bandwidth, and computational resources are typically non-conservable. An example of a resource that is conservable is battery power, as we are able to save it for future use. We are not claiming that the Buffet principle does not apply to conservable resources. But the principle seems more easily applied to non-conservable resources because applying it to conservable resources requires weighing current use against future use, which requires predictions of future behavior.

The second category is where the design controls the entire resource, as opposed to sharing the resource with Buffet-unaware users. The concern here is that non-Buffer users may not be able to distinguish between greedy usage, which has a lower marginal benefit, and normal usage. Such users might reduce their resource consumption on observing aggressive usage by Buffet users, which would reduce overall system throughput. This would be the case, for instance, if the transport protocol alluded to in Section 3 was implemented as is; the redundant packets sent by this new transport protocol would steal bandwidth from other users.

However, with careful design, it may be feasible for Buffet users to co-exist with non-Buffer users. For instance, coexistence is achieved in our wireless FEC design by making sure that the FEC bits are sent with a lower priority. A similar design would work for the transport protocol if we could send redundant packets at lower priority such that they are dropped by routers when resources are limited.

5.3 How much benefit does a Buffet design bring in practice?

It depends on the workload and the amount of resources. So it might vary from none to a lot. For example, consider our erasure coding system, in which the benefit can be quantified in terms of reduction in loss rate. Here, we observe zero loss when the incoming data rate is low to loss rate that is equal to the channel loss rate when the incoming data rate is more than the channel capacity. The appropriate way to view a Buffet design is that its performance is limited by the amount of spare available resources instead of being limited by specific design parameters. In other words, it can provide the best performance that a fixed level of resource investment can provide.

5.4 What are the side effects of greedily using resources (i.e., what is the fine print)?

We have encountered two side-effects. First, the performance of the system becomes a function of the workload itself. The system performs better under low load and exhibits poorer performance under higher loads. For instance, in a conventional FEC system, the observed loss rate is usually not a function of the workload; but in our Buffet FEC system, the observed loss rate does depend on the workload. One might argue that such a workload-dependent performance abstraction is hard for higher-layers. But observe that this already happens today in many instances. For instance, wireless loss rate changes with load because the rate of collisions changes. Similarly, even along a wired path, loss rate observed by applications can depend on the rate at which data is sent. Sending at 2 Mbps along a path with capacity 1 Mbps leads to a 50% loss rate, but sending at 3 Mbps leads to a 66% loss rate.

The second side-effect is that greedy usage may increase the latency of normal tasks in some situations. For instance, while the additional FEC bits are being transmitted, newly arrived data must wait. Similarly, new read requests may have to wait, while redundant blocks are being written to the disk. The increase in latency depends on the duration of the greedy task and whether it can be pre-empted. It can thus be ameliorated through pre-emption if possible and keeping individual greedy tasks small.

6. CONCLUSIONS

We proposed a different perspective on system design. This perspective advocates a more holistic view of available resources rather than blindly focusing on their efficient use. It can be summarized using the Buffet principle which states that we should continue using additional resources as long as the marginal benefit is more than the marginal cost of using the resources. Instead of being

limited by artificial design choices, such designs have the potential to enable the best performance the available level of resources can provide. Through several examples, we outline how Buffet designs differ from the more traditional efficiency-focused designs and how they are likely to perform significantly better.

We also discuss under what circumstances Buffet designs are likely to be most useful. Our limited experience points to both the strengths and challenges in incorporating the principle in practical designs. Overall, we find the principle to be promising and offering a useful perspective on system design. Its eventual worth can only be understood by studying the performance of concrete designs that are based on it. This is an active area of research for us.

7. REFERENCES

- [1] Medium access control (mac) quality of service enhancements. <http://standards.ieee.org/getieee802/download/802.11e-2005.pdf>, 2005.
- [2] J.-S. Ahn, S.-W. Hong, and J. Heidemann. An adaptive FEC code control algorithm for mobile wireless sensor networks. *Journal of Communications and Networks*, 7(4), 2005.
- [3] S. Ahn and A. Shankar. Adapting to route-demand and mobility in ad hoc network routing. *Computer Networks*, 38(6), 2002.
- [4] A. Levisianou, C. Assimakopoulos, N.-F. Pavlidou, and A. Polydoros. A recursive IR protocol for multi-carrier communications. In *Int. OFDM Workshop*, Sept. 2001.
- [5] M. Balakrishnan, T. Marjan, K. Birman, H. Weatherspoon, and E. Vulliamis. Maelstrom: Transparent error correction for lambda networks. In *NSDI*, Apr. 2008.
- [6] A. Balasubramanian, B. Levine, and A. Venkataramani. DTN routing as a resource allocation problem. Aug. 2007.
- [7] B. Esfahd. *Preload: An adaptive prefetching daemon*. PhD thesis, University of Toronto, 2006.
- [8] R. K. Guha, Y. Ling, and W. Chen. A light-weight location-aware position update scheme for high mobility networks. In *MILCOM*, Oct. 2007.
- [9] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE JSAC*, 16(3), 1998.
- [10] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *SAPIR*, Aug. 2004.
- [11] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *SIGCOMM*, Aug. 1996.
- [12] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. MAR: A commuter router infrastructure for the mobile Internet. In *MobiSys*, June 2004.
- [13] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN*, Aug. 2005.
- [14] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Performance analysis of mobility-assisted routing. In *MobiHoc*, May 2006.
- [15] Windows Vista SuperFetch. <http://www.microsoft.com/windows/products/windowsvista/features/details/superfetch.aspx>.
- [16] A. Venkataramani, R. Kokku, and M. Dahlin. TCP-Nice: A mechanism for background transfers. In *OSDI*, 2002.
- [17] Q. Yang and H. H. Zhang. Integrated web prefetching and caching using prediction models. *WWW*, 4(4), 2001.
- [18] L. Zhao, J. W. Mark, and Y. Yoon. A combined link adaptation and incremental redundancy protocol for enhanced data transmission. In *GLOBECOM*, Nov. 2001.