Energy Efficient Peer-to-Peer Storage

Geoffrey Lefebvre and Michael J. Feeley

Department of Computer Science
University of British Columbia

{geoffrey,feeley,}@cs.ubc.ca

Abstract

This paper describes key issues for building an energy efficient peer-to-peer (P2P) storage system. Current P2P systems waste large amounts of energy because of the false assumption that participating nodes' resources are free. Environmentally and economically, this is not true. Instead this paper argues that idle nodes in a P2P system should *sleep* to save energy. We derive an upper bound on the time an idle node can sleep without affecting the durability of the data stored in the system. This upper bound is parameterized by the replication factor and expected failure rates. We also outline a protocol for failure detection in an environment where only a small fraction of the nodes are alive at any time.

1 Introduction

In the last several years, many projects have emerged aimed at harnessing *idle* resources in large networks of desktop computers. Projects such as SETI@home and Project RC5 utilize idle machines to perform parallel computations on vast data sets. Similarly, peer-to-peer (P2P) systems such as CFS [6], PAST [16], Oceanstore/Pond [10, 13] and Farsite [1] confederate idle disk capacity into massively distributed data stores. The success of these ideas is rooted in a world in which idle network resources are plentiful, mainly because personal computers are actually used for only fraction of the day.

Unfortunately, idle resources are not free. A modern desktop computer consumes around 70W when idle [14] and easily over 125W when in use. In the year 2000, personal computers and workstations accounted for approximately 40% of the energy consumed by commercial office and telecommunication equipment in the United States [15]. Although this energy represented just over 1.0% of the total U.S. electrical consumption in that year, its generation produced 22 million metric tons of CO_2 and was valued at four billion US dollars (at a rate of 10 cents per KW-h). These figures exclude home computers and so the energy cost of computing was actually much higher.

Seen from this perspective, the current abundance of idle computing resources should perhaps be viewed, not as an opportunity to be exploited, but instead as a tremendous waste of environmental resources and money. A rough estimate of the potential scope of this waste can be calculated using results of recent surveys of computer use. In [19], for example, over 50% of computer and 30% of monitors surveyed were left powered on at night. Most of the computers that were powered down, were shut off manually; very few computers used automatic power management features to reduce their energy consumption when idle. In another case, a feasibility study for P2P storage at Microsoft [4] shows that a shockingly small 5% of the computers are shut down at night and only 10% during weekends. Based on these studies, and a conservative assumption that idle computers consume 70W of energy, which excludes their displays, we can estimate that in the year

2000, the 48 millions commercial computers in the US wasted 10–20 TWh of electricity, 5–10 million metric tons of CO_2 and 1–2 billion US dollars.

While this is a convincing argument for removing the power from idle computers, doing so threatens the viability of systems designed to exploit idleness. A naive, but economically and environmentally sound, approach of turning off machines when idle, would leave few idle resources to exploit. Perhaps there is room for a middle ground, in which computer power is used sensibly to provide both local and global services, but is not wasted to power machines when they are needed for neither purpose.

This paper examines this middle ground in the context of P2P storage. We begin with the observation that P2P systems waste energy by keeping machines powered even when they are needed neither by their owners nor to satisfy the current load demands of the P2P system. The main problem with many P2P storage system is that the departure of a node requires copying the data stored on that machine to other P2P nodes: a high cost. We describe a solution based on the idea of allowing P2P nodes to power down without first copying data to other machines. We describe several issues, chiefly that the system must differentiate between idle machines that will return to the P2P system and failed machines that will not. We show that by placing a bound on the amount of time an idle node sleeps before checking back in with the P2P system, the system can avoid confusing sleeping nodes from failed nodes. We provide a derivation for an upper bound that is parameterized by the number of nodes on which data is replicated and by assumptions of system failure behaviour.

2 Challenges

While methods such as processor scaling, display shutdown and disk spindown can be used to reduce energy consumption, they are not nearly as effective as turning the computer off or placing it in low-power hibernation [5]. An EEP2P system will thus be substantially more dynamic than current systems. Nodes will come and go frequently and will spend more time asleep and disconnected than awake. Node availability will also be bimodal, with a smaller fraction of nodes continuously failing or leaving the system permanently [2]. This dynamism presents an EEP2P system with challenges for durability, availability and inter-node communication, which we summarize in this section.

Durability characterizes the ability of a system to preserve data over time. P2P storage deals with failures by making data redundant using replication or encoding such as erasure codes[18]. Over time, nodes permanently leave the system and, if no counter-action is taken, the redundancy decays until data is permanently lost.

Two methods exist to maintain durability: failure monitoring and periodic refresh. Failure monitoring as in CFS [6] requires that nodes monitor each other for liveliness. When a node detects that another node has failed, it maintains redundancy by copying the failed node's data to a new node. In a EEP2P system, most node departures will be caused by idle node powering down and will thus be transient. If no effort is made to differentiate between these frequent transient departures and less frequent permanent ones, the required network bandwidth to maintain redundancy is prohibitively high [3]. This differentiation is key to the design of any P2P system that uses failure monitoring [2] and is even more critical for EEP2P systems.

Other systems such as OceanStore [10] use erasure codes as their redundancy mechanism. For the same storage overhead, erasure codes provide greater availability and durability than strict replication. To maintain durability, data is periodically reinserted into the system to restore the original redundancy. This refresh period must be frequent enough to ensure that neither durability nor availability is compromised.

Replication and erasure encoding can be combined to protect data with similar performance to erasure encoding alone. Using this combined approach, replication ensures durability and erasure encoding ensures availability. The system stores data by first splitting and encoding it to generate a sufficiently large number of fragments to ensure availability. Each fragment is then inserted independently into a replicated storage system that uses failure monitoring, not periodic refresh, for durability. This approach performs much better than replication alone, because it requires fewer replica nodes than needed to ensure that *all* of the original data is always available. We believe that it should also perform roughly as well as erasure encoding alone. Furthermore, clique-based replication has clustering behaviour lacking from pure erasure encoding; we believe thus clustering will be useful for various aspects of a EEP2P implementation. For these reasons, we focus on this combined approach for the rest of this paper. We acknowledge, however, that the utility of an alternative based on erasure encoding alone remains an open and interesting question.

The first challenge in the design of an EEP2P system is to differentiate between transient and permanent departures. To do so, it is sufficient to place a bound on how long an idle node is allowed to be disconnected from the system. Nodes that are disconnected longer than this time can thus be considered to have permanently departed. The upper bound is based on the replication factor and the estimated rate of permanent departures. Section 3 covers this upper bound in detail. We have also computed a lower bound on sleeping that is based on the time and energy cost for power-cycling a node and the possible damage that frequent power cycling can do to certain components such as disks. Typical values are around 100–200 s. Space limitations prevent further discussion of this lower bound.

The second challenge for EEP2P is handling communication when only a small fraction of nodes are available. How do nodes exchange information reliably when the probability of both a specific sender and receiver being awake at the same time is small? An important situation where this challenge arises is failure monitoring. In this case, a set of nodes monitors each other's liveliness using periodic message exchanges. Each node, however, may see only a small fraction of the non-failed nodes at a time. How does a node determine if some other node is alive when both may never be awake at the same time? Section 4 covers this topic in detail.

3 An Upper Bound on Sleeping

This section presents and explains a formula for $\tau_{s_{max}}$, the maximum time an idle node can sleep before it can be deemed to have permanently failed. An EEP2P node utilizes the standard idle-sleep mechanism available on most modern PCs with one exception. Before going to sleep, nodes program a wakeup time for $\tau_{s_{max}}$ in the future; programmed wakeup is also a standard feature on most PCs. When a node wakes, it remains awake for a minimum period of time as specified in Section 4. It then returns to sleep if it is still idle.

The main consideration for $\tau_{s_{max}}$ is that it must be small enough so that the system can detect permanent failures in time to re-replicate data before it is lost. It is thus a function of the system's replication factor and the expected node-failure rate.

As in other replication-based P2P systems such as CFS [6] and Chord [17], nodes operate in cliques of size k, where k is the replication factor, and monitor each other by sending periodic heartbeat messages. For ease of exposition, we initially assume that there exists a bulletin board where nodes can post and read messages. Any node that hasn't posted a note in the last $\tau_{s_{max}}$ is assumed to have permanently left the system. We remove this simplification in Section 4.

We also assume that time is divided into epochs of length $\tau_\varepsilon = \tau_{s_{max}} + \delta$ where $\delta \to 0$ such that any node that does not permanently depart during an epoch is guaranteed to be awake at least once during this epoch.

We observe that failures that occur in epoch ε_i will be detected before the end of epoch ε_{i+2}. To see why it can take this long, consider a node A that fails in epoch ε_i after posting to the bulletin board in that epoch. It is possible that all nodes that are alive in the following epoch, ε_{i+1}, are awake at a time that is less that $\tau_{s_{max}}$ from A's last bulletin-board posting; these nodes can not detect that A has failed. It is only in the next epoch, ε_{i+2}, that any node that checks the bulletin board is guaranteed to see that A has failed.

We can now see that as long as at least $\lceil \frac{k}{3} \rceil$ are alive at the beginning of any epoch and at most $\lceil \frac{k}{3} \rceil - 1$ fail in any epoch, there will always be at least one node alive in each epoch that will detect all failures that occurred two epochs in the past. If failed nodes can be replaced instantly[1] at the end of the epoch, the number of nodes alive at the start of the next epoch will be $\geq \lceil \frac{k}{3} \rceil$.

What is left is to determine τ_ε, the longest epoch that preserves the guarantee that at most $\lceil \frac{k}{3} \rceil - 1$ nodes fail during the epoch. To do this we use a Poisson model as in [11][12]. Failures are exponentially distributed with rate parameter μ. The probability that a node fails within time t is $1 - e^{-\mu t}$.

P, the probability that too many nodes fail in an epoch (i.e., $\lceil \frac{k}{3} \rceil$ or more) is given by:

$P = \sum_{i=\lceil \frac{k}{3} \rceil}^{k_s} \binom{k_s}{i} (1 - e^{-\mu \tau_\varepsilon})^i (e^{-\mu \tau_\varepsilon})^{k_s - i}$

where k_s is the number of nodes alive at the start of an epoch. To express our key constraint, that fewer than $\lceil \frac{k}{3} \rceil$ fail in an epoch w.h.p[2] we must ensure that $P < N^{-c}$ with $k_s = k$ in the worse case. Solving for τ_ε provides a bound on $\tau_{s_{max}}$ for a given replication factor k, failure rate μ and c the desired number of 9's for durability.

k	$c=6$	$c=8$	$c=10$
7	0.003	0.00066	0.00014
10	0.00843	0.00264	0.00083
16	0.0234	0.016	0.00486

The table above gives normalized values of $\tau_{s_{max}}$ for two different durability levels and three replication factors. A value for $\tau_{s_{max}}$ is computed by divided the reported value by μ. For example, [4] reports an expected lifetime$\frac{1}{\mu}$ of 290 days. In this case, for $k = 10$ and $c = 8$, $\tau_{s_{max}} = 18.4$ hours and for $k = 7$, $c = 8$, $\tau_{s_{max}} = 4.6$ hours.

4 Communication among Peers

Section 3 describes an upper bound on how long nodes can sleep based on the assumption of a hypothetical bulletin board. We now present a gossip-oriented communication mechanism that, for purposes of this algorithm, is equivalent to such a bulletin board.

We first define f to quantify the potential energy savings of the system. We ensure that all nodes are available at least $1/f$ of the time by requiring that idle nodes remain awake for $\tau_{s_{max}}/f$ each time they wake. This compromise simplifies failure detection, but increases the minimum power idle nodes consume to be $1/f$ of their normal idle power (e.g., 70/f W for a typical PC).

The basic scheme for implementing the shared bulletin board is as follows. Each node maintains a private timestamp vector with entries for each node in its clique. Nodes include their timestamp vector in their own heartbeat messages. When a node receives a heartbeat message it merges the message's vector with its own. Each node then uses their private timestamp vector exactly as they did in the shared bulletin board described previously. Nodes check their vector periodically for any node whose locally recorded timestamp value is more that $\tau_{s_{max}}$ in the past. They deem all such nodes as having permanently failed and thus immediately trigger re-replication.

In the case where less than two nodes per clique are up at the same time, some timestamp updates may not propagated to all nodes. This inconsistency can lead the system to falsely label a node as having permanently failed and thus unnecessarily re-replicate its data. It is possible to reduce the probability of these false positives by grouping node cliques into *super cliques* for purposes of failure detection. Doing so allows the system to use larger super cliques for failure detection without increasing the replication factor k.

The probability that at least two nodes of a size-x (i.e., $x > k$) super clique are up at all times is given by the following binomial.

$P = \sum_{i=2}^{x} \binom{x}{i} (\frac{1}{f})^i (1 - \frac{1}{f})^{x-i}$

It's easy to see that the ratio x/f must be large (e.g., ≥ 16) to guarantee with decent probability the propagation of information.

[1] Since in reality new nodes will be inserted into the system during an epoch, we feel this is a valid assumption

[2] In this paper w.h.p (with high probability) denotes probability $1 - N^{-c}$. To express this probability in terms of 9's, $N = 10$ and c represents the number of 9's.

5 Open Questions

Many open questions remains to be answered. For example, how does an EEP2P store deal with load variations? So far we've assumed the load was generated by peers so it is proportional to the number of non-idle nodes in the system. When most of the system is in low-power mode, the load should be low. This might not always be the case. It may be necessary, for example, to wake nodes remotely when needed or to predict future load when putting nodes to sleep so that sleep times are shorter when load is higher.

Another important question is whether caching can improve performance. Availability is defined as the fraction of time files are available in the system. If instead, availability were defined as the fraction of satisfied requests, perhaps caching popular files and keeping fewer replicas of non-popular files could reduce total storage overhead while maintaining the same level of availability.

Finally, in a worldwide system where nodes span the globe, if the system was designed as a hierarchical set of rings[7][9] organized by timezone, could caching within an individual time zone ring help concentrate the load on active nodes and thus reduce the load on idle nodes in other time zones. Could this lead to an increase in energy saving?

6 Related Work

There has been many projects on P2P storage. CNS [6], PAST [16], OceanStore/Pond [10, 13] and Farsite [1], to name a few, are all large scale decentralized storage but none of them so far have addressed the issue of power consumption.

The energy consumption problem has been looked at in other areas. Chase et al [5] proposes an architecture for power management in data hosting centers. Gupta et al [8] looks at the power consumption of networking devices such as switches and routers on the Internet.

7 Conclusion

This paper argues that idle nodes should be turned off or placed in low-power hibernation to conserve energy. We further argue that a P2P storage system should differentiate between powered off idle nodes and permanently departed nodes, thus avoiding the significant data-copy cost of re-replicating data stored on idle nodes. The key to this differentiation is to place an upper bound on the amount of time that nodes are allowed to sleep, deeeming nodes that are unavailable for longer than this bound to have permanently departed the system. We presented a formula for this bound and show that typical values fall within the range of four to eighteen hours. We also presented a gossip-oriented communication protocol for detecting node failure that allows idle nodes to consume a fraction $1/f$ of the energy the would if they were always powered on.

References

[1] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002.

[2] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proc. of IPTPS '03*, 2003.

[3] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *HotOS IX*, 2003.

[4] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proc of SIGMETRICS*, 2000.

[5] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc of SOSP*, 2001.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *In Proc. of SOSP*, 2001.

[7] M.J. Freedman and D. Mazieres. Sloppy hashing and self-organizing clusters. In *Proc. of IPTPS '03*, 2003.

[8] M. Gupta and S. Singh. Greening on the internet. In *Proc. of SIGCOMM*, 2003.

[9] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. of USITS '03*, 2003.

[10] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proc. of ASPLOS*, 2000.

[11] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proc. of PODC*, 2002.

[12] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. In *Proc. of FOCS*, 2001.

[13] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proc. of USENIX FAST*, 2003.

[14] Judy A. Roberson, Gregory K. Homan, Akshay Mahajan, Bruce Nordman, Carrie A. Webber, Richard E. Brown, Marla McWhinney, and Jonathan G. Koomey. Energy use and power levels in new monitors and personal computers. In *Energy Analysis Department, Environmental Energy Technologies Division, Ernest Orlando Lawrence Berkeley National Laborratory, University of California*, 2002.

[15] Kurt Roth, Fred Goldstein, and Jonathan Kleinman. Energy consumption by commercial office and telecommunications equipment in the u.s., 2002.

[16] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of SOSP*, 2001.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001.

[18] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS '02*, 2002.

[19] C. A. Webber, J. A. Roberson, R. E. Brown, C. T. Payne, Bruce Nordman, and Jonathan G. Koomey. Field surveys of office equipment operating patterns, 2001.