BlockParty: Cooperative offsite backup among friends

Tran Dinh Nguyen, Jinyang Li

New York University
{trandinh, jinyang}@cs.nyu.edu

We are keeping an increasing amount of valuable data in digital format only. Offsite backup systems are ideal for ensuring that such irreplaceable personal information will be preserved in the face of disk failures, unexpected disastrous events and operator mistakes. It is time to bring offsite backup solutions, traditionally a luxury to only deep pocketed enterprises, to the masses of home users today.

There are two difficulties in building a cooperative backup system on a traditional peer-to-peer network. First, there is not enough bandwidth to repair lost data due to a high node departure rate [2]. The presence of frequent node departures is a consequence of users failing to display long term cooperativeness without proper incentives: after downloading a file, users have no good reason to continue participating in the network. Second, unlike file sharing, we cannot induce cooperative behavior in backup systems using tit-for-tat [3, 4]. Tit-for-tat requires that both peers involved an exchange have some resource that catches the other’s fancy. However, nodes that seek to restore data from others following a disk crash have no useful data for other peers and therefore are inevitably at the mercy of others’ altruism.

Most existing peer-to-peer storage systems run under a centralized administration [1] so that any detected misbehavior or unwillingness to cooperate can potentially result in a peer’s expulsion from the system. Such a trusted central authority can be a serious nuance in situations when there is no natural central party trusted by all users. In BlockParty, a node’s owner explicitly specify a small number of over-lay neighbors to perform cooperative backups with. Nodes that agree out-of-hand to cooperate with each other are likely to be owned by users who have real-world friendships among them. In essence, each BlockParty node acts as its own centralized membership authority.

BlockParty’s particular system structure brings us advantages as well as challenges. From a resource donor’s point of view, the ability to explicitly specify which other nodes she contributes her resources to gives a node’s owner better incentive to participate in the system over a long time and increase her node’s availability to others. From a resource recipient’s point of view, entrusting data only to those nodes’ whose owners are trusted “friends” allows each node to act as its own trusted authority to deter others from misbehaving and denying services with such credible threats that is the ire of friends in real life. For such credible threats to be effective, BlockParty cannot rely on transitive trust [5, 7], i.e. nodes never store data on those that they do not have a direct real world relationship with. We do not think this is overly restrictive because unlike peer-to-peer file sharing and live streaming applications, there is little value in making use a huge set of unreliable nodes to store one’s backup data.

Restricting where data is stored limits BlockParty’s ability to utilize idle disk spaces in the system effectively. It is common that a node could not back up her data fully because none of its neighbors has any idle disk space remaining. BlockParty attacks this resource imbalance problem by having overloaded nodes store coded data blocks that belong to different neighbors, effectively increasing an overloaded node’s ability to store more information without literally requiring it to contribute more disk spaces. Although a node trusts its neighbors not to purposefully deny service at the time of restore, it still verifies the presence of its backed up data at neighbors periodically to be able to detect data loss and start the repair process promptly. The verification protocol must only involve transferring a small amount of data between nodes. Furthermore, it must accommodate coded information belonging to multiple nodes. BlockParty uses homomorphic hash functions [6] to allow a node to compute a secure hash value of any individual block from the hashes of the aggregate coded block and the hashes of remaining individual blocks.

References


