Efficient Replica Maintenance for Distributed Storage Systems

The area of focus of the Carbonite algorithm is durably and cost-efficiently storing of immutable object in a system that aggregates the disks of many Internet nodes. Hence, Carbonite is a replication algorithm for large storage systems. More.

To ensure availability (immediate access to data) of data, replication systems responds to disk failure by creating new replicas. When availability is key, replication is immediately used regardless if the failure was a disk failure or just a transient (temporary) failure. The problem of this approach is that constantly making new replicas is costly, and do not scale to large systems. The developers of Carbonite realized that Internet users can tolerate some unavailability as long as they eventually will be able to view what they requested. This knowledge is the main motivation behind Carbonite. With main focus on durability, not availability, they could build an Internet scale replication system. Need to talk about reintegration and replication faster than failure, but I don't know if it should be done here or in the technique-paragraph.

As mentioned above, being able to maintain large storage application systems require sophisticated techniques for when and how replication is performed. Carbonite only makes replications in respond to failure if the number of replicas for a given object is below the threshould r_L, where r_L is the number of replicas needed to survive a burst of failures. This property is enforced by re-integration of failed nodes when they come back up again. By remembering which replicas was stored on a node, they can be reused, and thereby make sure that replication only is performed when the number of replications is below r_L, this technique drastically reduces the number of replications required. Another technique to reduce the number of replications made as responds to transient failures is timeouts. By waiting a given amount of time before creating a new replica, the node might come back, and cancel the planned replication.