BitTorrent: Strategies

Piece Downloading Strategy

FOUNDATION OF PEER-TO-PEER SYSTEMS

- Clients may choose to download pieces in random order.
- *A better strategy is to download pieces in* rarest first *order.*

QUIZ: How to determine the rarest pieces?

The client can determine this by keeping the initial bitfield from each peer, and updating it with every 'have *message. Then, the client can download the pieces that appear least frequently in these peer bitfields. Note that any Rarest First strategy should include randomization among at least several of the least common pieces, as having many clients all attempting to jump on the same "least common" piece would be counter productive.*

End Game Strategy

- When a download is almost complete, there's a tendency for the last few blocks to trickle in slowly. To speed this up, the client sends requests for all of its missing blocks to all of its peers. To keep this from becoming horribly inefficient, the client also sends a cancel to everyone else every time a block arrives.

- *When to enter end game mode? (open area of discussion).*
  - *Some clients enter end game when all pieces have been requested. Others wait until the number of blocks left is lower than the number of blocks in transit, and no more than 20.*
  - *There seems to be agreement that it's a good idea to keep the number of pending blocks low (1 or 2 blocks) to minimize the overhead,*
  - *and if you randomize the blocks requested, there's a lower chance of downloading duplicates.*

QUIZ: Why BitTorrent slows down near the end of downloading a file?

Choking & Unchoking Strategy

- All peer has limited connection capability. Each peer need to use tit-for-tatish strategies to ensure that they get a consistent download rate.

- Some criteria a good choking algorithm should meet:
  - Cap the number of simultaneous uploads for good TCP performance.
  - Avoid choking and unchoking quickly or 'fibrillation'.
  - Reciprocate to peers who let it download.
  - Try new peers once in a while to find better peers.

- By definition any peer may use any strategy.

A Sample Choking Unchoking Strategy

- Avoids fibrillation by only changing choked peers once every ten seconds.
- Reciprocation and number of uploads capping is managed by unchoking the four peers which have the best upload rate and are interested. (This maximizes the client's download rate. These four peers are referred to as *downloaders*, because they are interested in downloading from the client.)
- If there is not enough ‘interested’ peers then the ‘not interested’ peers which have a better upload rate (as compared to the *downloaders*) get unchoked.
- If any in the second group become interested, the *downloader* with the worst upload rate gets choked, and this peer is moved into the downloader set.
- Once a client obtains a complete file, it uses its upload rate rather than its download rate to decide which peers to unchoke.
-
- (continued..)

Optimistic Unchoking Strategy

- The peer also keeps one connection available for searching better peers. This is called ‘optimistic unchoking’.
- For optimistic unchoking, at any one time there is a single peer which is unchoked regardless of its upload rate (if interested, it counts as one of the four allowed *downloaders*).
- Which peer is optimistically unchoked rotates every 30 seconds.
- Newly connected peers are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation. This gives them a decent chance of getting a complete piece to upload.

Anti-snubbing

- Occasionally a BitTorrent peer will be choked by all peers which it was formerly downloading from. In such cases it will usually continue to get poor download rates until the optimistic unchoke finds better peers.
- To mitigate this problem, when over a minute goes by without getting any piece data while downloading from a peer, BitTorrent assumes it is "snubbed" by that peer and doesn't upload to it except as an optimistic unchoke.
- This frequently results in more than one concurrent optimistic unchoke, (an exception to the exactly one optimistic unchoke rule mentioned above), which causes download rates to recover much more quickly when they falter.

Request Queuing

- In general peers are advised to keep a few unfullfilled requests on each connection. This is done because otherwise a full round trip is required from the download of one block to begining the download of a new block (round trip between PIECE message and next REQUEST message). On links with high BDP (bandwidth-delay-product, high latency or high bandwidth), this can result in a substantial performance loss.
- *Implementer's note: This the '*most crucial performance item. *A static queue of 10 requests is reasonable for 16KB blocks on a 5mbps link with 50ms latency. Links with greater bandwidth are becoming very common so UI designers are urged to make this readily available for changing. Notably cable modems were known for traffic policing and increasing this might of aleviated some of the problems caused by this.*
- ***Compute based on TCP window Rule (research!)***
- ***View #2*** NOTE: The "defaults to 5 outstanding requests" hasn't been true for a long time, "32 KB blocks" is misleading since you normally don't use 32 KB blocks, and tuning queue length by changing it and trying to measure the effects is a bad idea.

Super Seeding (experimental)

- The super-seed feature is a new seeding algorithm designed to help a torrent initiator with limited bandwidth "pump up" a large torrent, reducing the amount of data it needs to upload in order to spawn new seeds in the torrent.
- When a seeding client enters "super-seed mode", it will not act as a standard seed, but masquerades as a normal client with no data.
- As clients connect, it will then inform them that it received a piece -- a piece that was never sent, or if all pieces were already sent, is very rare. This will induce the client to attempt to download only that piece.
- When the client has finished downloading the piece, the seed will not inform it of any other pieces until it has seen the piece it had sent previously present on at least one other client.
- Until then, the client will not have access to any of the other pieces of the seed, and therefore will not waste the seed's bandwidth.

Super Seeding (continued..)

- This method has resulted in much higher seeding efficiencies by:
  – Inducing peers into taking only the rarest data,
  – Reducing the amount of redundant data sent,
  – Limiting the amount of data sent to peers which do not contribute
- Prior to this, a seed might have to upload 150% to 200% of the total size of a torrent before other clients became seeds. However, a large torrent seeded with a single client running in super-seed mode was able to do so after only uploading 105% of the data. This is 150-200% more efficient than when using a standard seed.
- Super-seed mode is 'NOT recommended for general use. While it does assist in the wider distribution of rare data, because it limits the selection of pieces a client can download, it also limits the ability of those clients to download data for pieces they have already partially retrieved. Therefore, super-seed mode is only recommended for initial seeding servers.

BitTorrent: Smart Extensions

Extension Ideas?

- What's the usual problems?
  - Large messages (overhead).
  - Indefinite waits
  - Slow start for downloaders
  - Slow start for seeders.
  - Slow finish.
  - Not a realP2P?
- A set of extensions known as fast BitTorrent allow a peer to more quickly bootstrap into a swarm by giving a peer a specific set of pieces which they will be allowed to download regardless of choked status and reduce message overhead.

FOUNDATION OF PEER-TO-PEER SYSTEMS

LECT-01, S-50
IAD99S, javed@kent.edu
Javed I. Khan@1999

Fast Extensions

- The Fast Extension modifies the semantics of the *Request*, *Choke*, *Unchoke*, and *Cancel* messages.
- It also adds
  - **Have All/Have None**
  - **Suggest Piece**
  - **Reject Request**
  - **Allowed Fast Set Generation**
- Now, every request is guaranteed to result in EXACTLY ONE response which is either the corresponding reject or corresponding piece message.
- The specificication is documented at the BitTorrent site here: http://www.bittorrent.org/fast_extensions.html

Full or Empty Bitfield

- **Have All/Have None**
- *Have All*: <len=0x0001><op=0x0E> *Have None*: <len=0x0001><op=0x0F>
- *Have All* and *Have None* specify that the message sender has all or none of the pieces respectively. When present, *Have All* or *Have None* replace the *Have Bitfield.*
- Exactly one of *Have All*, *Have None*, or *Have Bitfield* MUST appear and only immediately after the handshake.
- The reason for these messages is to save bandwidth. Also slightly to remove the idiosyncrasy of sending no message when a peer has no pieces.
- When the fast extension is disabled, if a peer receives *Have All* or *Have None* then the peer MUST close the connection.

Explicit Rejection

- **Reject Request**
- *Reject Request*: <len=0x000D><op=0x10><index><begin><offset>

- Explicitly notifies a requesting peer that its request will not be satisfied. Now every request is matched by an explicit reply.
  - If a peer receives a reject for a request that was never sent then the peer SHOULD close the connection.
  - If a peer sends a choke, it MUST reject all requests from the peer to whom the choke was sent except it SHOULD NOT reject requests for pieces that are in the *allowed fast set.*
  - A peer SHOULD choke first and then reject requests so that the peer receiving the choke does not re-request the pieces.
  - If a peer receives a request from a peer its choking, the peer receiving the request SHOULD send a reject unless the piece is in the *allowed fast set.*

Super Seeding

- **Suggest Piece**
- *Suggest Piece*: <len=0x0005><op=0x0D><index> *Suggest*

- *Piece* is an advisory message meaning "you might like to download this piece." The intent for 'super-seeding' is to infuse pieces into network strategically (for various objectives).

  - Avoid throughput reduction, to avoid redundant downloads, and so that a seed which is disk I/O bound can upload contiguous or identical pieces to avoid excessive disk seeks.

  - The seed SHOULD operate to maintain a roughly equal number of copies of each piece in the network.

- A peer MAY send more than one *suggest piece* message at any given time. A peer receiving multiple *suggest piece* messages MAY interpret this as meaning that all of the suggested pieces are equally appropriate.

Fast Initiation in Swarm

- **Allowed Fast**
- *Allowed Fast: <len=0x0005><op=0x11><index>*
- New peers take several minutes to ramp up before they can effectively engage in BitTorrent's tit-for-tat. The reason is simple: starting peers have few pieces to trade.
- *Allowed Fast* is an advisory message which means "if you ask for this piece, I'll give it to you even if you're choked." *Allowed Fast* thus shortens the awkward stage during which the peer obtains occasional optimistic unchokes but cannot sufficiently reciprocate to remain unchoked.
- The *allowed fast set* is generated using a canonical algorithm that produces piece indices unique to the message receiver so that if two peers offer *k* pieces fast it will be the same *k*, and if one offers *k+1* it will be the same *k* plus one more.
- *k* should be small enough to avoid abuse, but large enough to ramp up tit-for-tat (currently set *k* to 10, but peers are free to change).
- *Warning! The message sender MAY list pieces that the message sender have not downloaded yet.*

FOUNDATION OF PEER-TO-PEER SYSTEMS

LECT-01, S-55
IAD99S, javed@kent.edu
Javed I. Khan@1999

Distributed Hash Table

- This extension is to allow for the tracking of peers downloading torrents without the use of a standard tracker.
- A peer implementing this protocol becomes a "tracker" and stores lists of other nodes/peers which can be used to locate new peers.
- The specification is documented at the BitTorrent site here: http://www.bittorrent.org/Draft_DHT_protocol.html

FOUNDATION OF PEER-TO-PEER SYSTEMS

LECT-01, S-56
IAD99S, javed@kent.edu
Javed I. Khan@1999

A Performance Graph

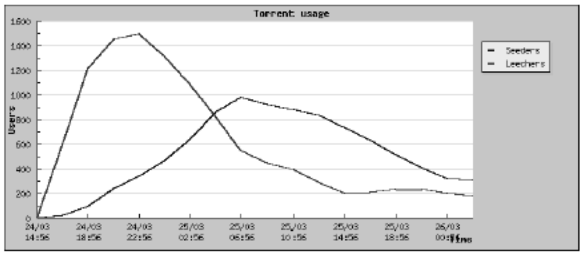

Figure 1: The number of complete downloaders ('seeders') and incomplete downloaders ('leechers') of a large deployment of an over 400 megabyte file over time. There must have been at least 1000 successful downloads, since at one time there were that many complete downloaders. The actual number of downloads completed during this period was probably several times that.

FOUNDATION OF PEER-TO-PEER SYSTEMS

LECT-01, S-57
IAD99S, javed@kent.edu
Javed I. Khan@1999

Next Class: Gnutella