

CONTENTS

第1章 はじめに	1
1.1 本書の対象と目的	2
1.2 本書の構成	7
1.3 下準備	8
第2章 配信の基礎	9
2.1 配信のとらえ方	11
2.1.1 配信の根幹	12
2.2 標準仕様でやりとりする	13
Column RFCのMUST/SHOULD—RFC 2119	15
2.2.1 配信の原則	15
2.3 配信の経路	16
2.3.1 ブラウザでWebページを表示するまで	17
2.3.2 インターネットアクセス時の経路を体験する	19
Column tracerouteで前後するRTT	24
Column 帯域・通信速度・レイテンシ	24
Column 帯域と通信速度の違いを理解する	25
Column プロトコルによるレイテンシへの影響	26
Column 5Gはどこを高速化するのか	27
2.4 配信をより高速なものにするために	27
2.4.1 配信最適化のためになにができるのか	28
2.4.2 配信経路の最適化	28
2.4.3 クライアントとサーバーでの最適化	29
Column Webサイトのチューニング	31
2.5 キャッシュの格納場所による分類	31
2.5.1 クライアントのローカルキャッシュ	32
2.5.2 経路上のキャッシュ	33
Column CDNの割り振りのしくみ	33
Column EDNS Client Subnet (ECS) —RFC 7871	35
2.5.3 ゲートウェイ (サーバー/オリジン側) のキャッシュ	36
2.6 private/sharedキャッシュ	38
2.6.1 キャッシュの位置とprivate/shared	39
2.6.2 private/sharedキャッシュの注意点	41
2.7 どこでどうキャッシュすべきか	43

2.7.1	キャッシュは誰が管理しているのか	44
Column	CDNとゲートウェイキャッシュの違い	45
第3章	HTTPヘッダ・設定とコンテンツの見直し	47
3.1	HTTPヘッダの重要性	49
3.1.1	実際のサイトのヘッダを見る	50
3.1.2	ヘッダとどう付き合うべきか	55
Column	Apacheでのヘッダ操作	57
3.2	HTTPメッセージ	59
3.2.1	開始行—RFC 7230#3.1	60
3.2.2	ヘッダ—RFC 7230#3.2	61
3.2.3	ボディ—RFC 7230#3.3	63
3.2.4	HTTP/2でのHTTPメッセージ	63
3.3	ステータスコードと説明句—RFC 7231#6	65
Column	暫定応答の1xxと最終応答の1xx以外	67
3.3.1	代表的なステータスコード	68
Column	Status Code 418	73
3.4	HTTPとキャッシュ	74
3.4.1	キャッシュを行う・使う条件	76
3.5	Cache-Controlによるキャッシュ管理	81
Column	リクエスト時にもCache-Controlは送信される	82
3.5.1	キャッシュの保存方法を指定 (未指定/public/private)	82
Column	キャッシュの仕様改定	85
3.5.2	キャッシュの使われ方を指定する (no-store, no-cache)	87
3.5.3	キャッシュの更新の方法を指定する (must-revalidate, proxy-revalidate, immutable)	88
3.5.4	オブジェクトの取り扱いを指定する (no-transform)	90
Column	中間でのデータ変更 (データサーバー・通信の最適化)	90
3.6	Cache-Controlにおける期限指定	91
3.6.1	キャッシュの期限と状態	91
3.6.2	キャッシュは指定した期間保存されるとも限らない	92
3.6.3	期限が切れたからといってすぐに消されない	92
3.6.4	キャッシュの期限指定 (max-age)	93
3.6.5	経路上の期限指定 (s-maxage)	94
3.6.6	Stale キャッシュの利用方法を指定する (stale-while-revalidate/stale-if-error)	94
3.7	Cache-ControlとExpiresヘッダ	95

3.8	TTLが未定義時の挙動—RFC 7234#4.2.2	95
3.9	キャッシュをさせたくない場合のCache-Control	96
	Column 誤ったmax-ageの指定	97
	Column キャッシュ不可なステータスコードをキャッシュする	98
	Column 仕様はすべて実装されているとは限らない	99
3.10	さまざまなリクエスト	100
	3.10.1 部分取得リクエスト (Range) —RFC 7233	100
	3.10.2 条件付きリクエスト (If-Modified-Sice/If-None-Match) —RFC 7232	100
3.11	さまざまなヘッダ	105
	3.11.1 Vary—RFC 7234#4.1	105
	3.11.2 Content-Type	108
	Column 拡張子とMIMEタイプ	110
	Column HTTP/2は使わなくてははいけないのか	110
	Column HTTPのセマンティクスとRFC	112
	Column HTTPを使う上でのベストプラクティスを紹介するRFC	113
3.12	HTTPヘッダの不適切な設定で起きた事例	114
	3.12.1 Cache-Controlが未定義	114
	3.12.2 コンテンツが更新されていないにもかかわらずETagが変わる	115
	Column ロードバランサーとロードバランシング	116
3.13	ヘッダクレンジング	116
	3.13.1 ゲートウェイとオリジンサーバーを把握する	116
	3.13.2 どのようにコントロールするのか	118
3.14	コンテンツのサイズ削減	120
	3.14.1 テキストが圧縮転送されていない	120
	Column どこで圧縮するべきか	121
	3.14.2 ストレージサービスの圧縮漏れ	122
	Column 圧縮は万能ではない	122
3.15	適切なメディアの選択によるコンテンツの改善	123
	3.15.1 配信システムとファイルサイズ	123
	3.15.2 画像サイズが必要以上に大きくないか (サムネイル)	124
	3.15.3 bppを考える	127
	3.15.4 画像フォーマットは適切か	128
	Column WebPと画像サイズ削減	132
	Column JPEG画像の注意点	136
	Column 外部サービスという選択肢	137
3.16	問題点を調査する	138

3.16.1	圧縮転送が有効になっていない	138
3.16.2	ETag/Last-Modifiedがおかしい	139
3.16.3	画像コンテンツが大きい	141
3.16.4	キャッシュが有効に使われていない	142
第4章	キャッシュによる負荷対策	143
4.1	キャッシュの構成・設定例	145
4.2	さまざまな負荷とその事例	147
4.3	負荷をさばくこととキャッシュ	150
4.3.1	キャッシュ導入でどの程度パフォーマンスが向上したか	151
4.4	キャッシュを使わない場合どうさばくか	152
4.5	キャッシュを使う	153
	Column ProxyやCDNはなぜ大量のリクエストをさばけるのか	154
4.5.1	流入が増えるケースを追って考える	155
4.5.2	キャッシュの役割	158
4.5.3	静的ファイルの配信でもキャッシュは有効	161
4.6	キャッシュ事故を防ぐ	163
4.6.1	中間 (Proxy/CDN) を信頼しすぎる	163
4.6.2	オリジンを信頼しすぎる	165
4.7	キャッシュ戦略・キャッシュキー戦略	165
4.7.1	クライアントからのリクエストを実際に組み合わせる	169
4.7.2	キャッシュキーとVaryのどちらに設定するか	178
4.7.3	キャッシュフレンドリなパス設計	184
4.8	キャッシュのTTLとキャッシュを消す	186
4.9	キャッシュのTTL計算と再利用—RFC 7234#4.2	186
	Column 生成時刻とTTLを決める際の基準	189
4.9.1	キャッシュの状態とStale	189
4.9.2	TTL=0 (max-age=0) とは何か	193
	Column max-age=0とno-cacheの違い	194
4.9.3	TTLの決め方	194
4.9.4	もうひとつのキャッシュ エラーキャッシュ	199
4.10	キャッシュの消去	199
4.10.1	無効化と削除	200
4.10.2	キャッシュ消去の注意点	201
4.10.3	キャッシュ消去にどこまで依存するか	202
4.10.4	サロゲートキー・キャッシュタグ	203

第5章	より効果的・大規模な配信とキャッシュ	205
5.1	<u>ヒット率だけによらない効率的なキャッシュの考え方</u>	207
	Column キャッシュアルゴリズム (LRU/LFU)	208
5.2	<u>どのようにリクエストを処理してレスポンスをするのか</u>	209
5.2.1	Proxy/CDNの機能を整理する	209
5.3	<u>RxReq—クライアントからのリクエストを受信する</u>	214
5.3.1	ACLの処理	214
5.3.2	キャッシュキーの操作およびVaryで指定されたセカンダリキーに関連するヘッダの操作	215
5.3.3	リクエストに含まれる情報に対する操作	215
5.3.4	キャッシュ可否の判定	216
5.4	<u>Cache lookup—キャッシュヒットの判定を行う</u>	217
5.5	<u>Wait for cache—キャッシュができるまで待つ</u>	218
5.6	<u>TxReq—オリジンにリクエストを送信する</u>	220
5.6.1	複数のオリジンがある場合の選択	220
5.6.2	キャッシュキーに影響を与えないオリジン問い合わせ時のホスト名・パス変更	221
5.6.3	イベントの呼び出し数を減らしたい場合のヘッダ操作	221
5.7	<u>RxResp—オリジンからレスポンスを受信する</u>	221
5.7.1	キャッシュ可否の判定 (2回目)	222
5.7.2	TTLを設定する	222
5.7.3	キャッシュするオブジェクトに対する操作	222
5.7.4	キャッシュヒット判定に関係するVaryヘッダの変更	223
5.8	<u>TxResp—クライアントにレスポンスを送信する</u>	223
5.8.1	キャッシュは変わらないがクライアントごとにヘッダを変える必要がある (CORS など)	223
5.8.2	不要なヘッダの編集	224
5.9	<u>キャッシュキーをVaryで代用する</u>	225
5.9.1	Varyの利用例	226
5.9.2	クライアントからの情報でセカンダリキー設定 (RxReq)	227
5.9.3	パスを変更する (TxReq)	228
5.9.4	キャッシュに格納されるVaryを変更する (RxResp)	228
5.9.5	クライアントに送るVaryに変更する (TxResp)	228
5.9.6	Varyもキャッシュキーも編集できないときのクエリ文字列	229
5.9.7	同一URLで複数のキャッシュを持つ方法の比較	230
5.10	<u>効果を高める対策</u>	233

5.10.1	ヒット率を上げる	233
Column	意図せず入る Vary	233
5.10.2	キャッシュするものを適切に選ぶ	240
5.10.3	ローカルキャッシュがある状態で即時のコンテンツ更新 (Cache Busting / no-cache)	241
5.10.4	操作する場所を意識する	243
5.11	動的コンテンツのキャッシュ	244
5.11.1	動的コンテンツとは何か	244
5.11.2	静的コンテンツ	245
5.11.3	動的コンテンツ	246
5.11.4	なぜ明確に TTL が決められないのか	247
5.12	分割してキャッシュする	249
5.12.1	キャッシュできるものとできないものを混ぜない	250
5.12.2	異なるライフサイクルのものはなるべく混ぜない	251
5.12.3	分離が困難だがとにかくキャッシュがしたい	251
5.12.4	API などコードで生成されたコンテンツのキャッシュ	251
5.13	Edge Side Includes (ESI)	253
5.13.1	ESI のメリット	256
5.13.2	ESI のしくみ	257
5.13.3	ESI のデメリット	259
5.13.4	応用的な使い方	260
5.14	配信構成の工夫	261
5.14.1	ストレージ	261
5.14.2	Proxy の増設 (スケールアウト)	264
5.15	多段 Proxy	265
5.15.1	Proxy を増設すると何が起きるのか	265
5.15.2	多段 Proxy のメリット——貫性	266
5.15.3	1 段方式の多段 Proxy	267
5.15.4	2 段方式の多段 Proxy	268
Column	ハッシュをとってどのように振り分けを行うか	269
5.15.5	どの方式の多段 Proxy を採用すべきか	269
5.15.6	多段 Proxy のメリットをより深く理解する	271
5.15.7	多段 Proxy の注意点	272
Column	キャッシュ巻き戻りの問題	277
Column	max-age が変動する時の注意	278
5.16	障害時に正しくサーバーを切り離す (ヘルスチェック)	279
5.16.1	ヘルスチェックを行う側の設定	279

5.16.2	ヘルスチェックを受ける側の設定	283
5.17	ドメインの分割	287
5.17.1	動的コンテンツと静的コンテンツでドメインを分ける	287
5.17.2	ZoneApex 問題への対策	288
第6章	CDNを活用する	291
6.1	なぜ CDN が必要なのかー自前主義だけだと難しい	293
6.1.1	自前主義がもたらす高コストと CDN	294
6.1.2	CDN + 自社配信という選択肢	295
6.2	CDNを使う前に	297
6.2.1	CDN とは	297
Column	オブジェクトストレージと CDN	300
6.2.2	CDN だからできること	301
Column	コラム ネットワークの品質が良いということはどういうことか	304
6.3	CDNの選び方	307
6.3.1	代表的な CDN の一覧	308
6.3.2	どの地域に配信をするのか (海外・国内配信)	309
6.3.3	ピーク帯域がどの程度あるのか CDN が対応できるか	309
6.3.4	どのようにキャッシュを制御するのか	310
6.3.5	HTTPS の取り扱い	311
6.3.6	CDN のキャッシュの消去	312
6.3.7	Apex ドメインの扱い	312
6.3.8	信頼できるのか	313
6.3.9	CDN の多段キャッシュ	313
Column	コンテンツの一貫性はどれほど必要なのか	318
6.4	CDNを使うときに気をつけたいポイント	319
6.4.1	キャッシュされない設定を調べる	319
6.4.2	クエリ文字列の解釈について	320
6.4.3	トラフィックなどの制限に注意	320
6.4.4	コンテンツのサイズに注意	321
6.4.5	Vary に注意	321
Column	キャッシュ汚染 DoS (CPDoS)	322
6.4.6	CDN のデバッグ	322
Column	CDN のヘッダと標準化	323
6.4.7	CDN で隠したい情報をきちんと整理する	324
6.4.8	ウォームアップ	326

6.5	クライアントの近くでコードを動かす - エッジコンピューティング	327
	Column 内部向けにCDNを使う - internal CDN	329
	Column CDNのコスト	329
6.6	CDNと障害	330
6.6.1	CDNの障害	330
6.6.2	問題の切り分け—それはCDNの障害なのか	331
6.6.3	まずはmetricsを見てみる	331
6.6.4	CDNの典型的な障害	332
6.6.5	コンテンツのダウンロードが遅い	337
6.6.6	キャッシュが壊れる	338
6.7	動的コンテンツのキャッシュやCDN利用は危険なのか	340
	Column マルチCDN	342
	Column DNSブロッキングにかかる	343
	Column 適切に設定しているのにキャッシュされない	344
6.8	実際にCDNを設定する	345
6.8.1	動的コンテンツの設定例	350
 第7章 自作CDN (DIY-CDN)		353
7.1	なぜCDNをつくるのか	354
7.1.1	低予算	355
7.1.2	ハイブリッドで使う	356
	Column CDN自体を多段で使うときの注意	358
7.2	低予算自作CDNの構成	360
7.2.1	さまざまなことを諦める	360
7.2.2	基本構成	361
	Column VPSのプランをどう選ぶか	366
7.3	自作CDNと外部CDNのハイブリッド構成	366
	Column 低コスト運用からその先へ	369
7.4	VCLでの設定例	369
7.4.1	gatewayとcacheでの設定の目的	369
7.4.2	リクエストの正規化	370
7.4.3	キャッシュする場合のCookieなどの取り扱い	371
7.4.4	TTLの設定	375
7.4.5	オリジンへの振り分け	379
7.4.6	ストレージの分割	382
7.4.7	オリジンでの注意	383
7.4.8	VCLサンプル	384

Appendix Varnishについて	393
<u>A.1 Varnishのインストール</u>	395
<u>A.2 読みたい公式ドキュメント</u>	395
<u>A.3 Varnishのサポート体制</u>	400
<u>A.4 基本的なVCL</u>	401
A.4.1 Varnishの起動方法と最小のVCLと考え方の基礎	401
A.4.2 文法の初歩	404
A.4.3 backendとdirector	405
A.4.4 サブルーチンとVCL変数	407
A.4.5 条件分岐・演算子・正規表現	411
A.4.6 VCLのデータ型と型変換	412
A.4.7 VCLを学ぶ	414
<u>A.5 VCLを記述する際の注意点</u>	414
A.5.1 "を含む文字列の指定方法	414
A.5.2 正規表現の取り扱い	414
A.5.3 デフォルトのVCLと同一名のVCLイベントの定義	415
A.5.4 VMOD	417
A.5.5 折り畳みをしてはいけないヘッダの扱い	417
<u>A.6 テストの重要性—varnishtest</u>	418
<u>A.7 varnishのログと絞り込み方</u>	421
A.7.1 VSLのグループ化	422
A.7.2 ログの絞り込み	424
<u>A.8 そのほかのツールやコマンド</u>	429
A.8.1 varnishstat	429
A.8.2 varnishadm	430
A.8.3 varnishtop	432
Column graceとkeepの取り扱い	433
Column VarnishとHTTPS	434
おわりに	435
参考文献	436
索引	437

はじめに

1.1 本書の対象と目的

本書はHTTPを介した配信とその最適化について解説する入門書です。配信に従事した経験のないアプリケーション・インフラエンジニアでもわかるように、初歩から説明していきます。

配信と言われても、言葉の意味も広く、何が何だか想像もつかないという方もいるかもしれません。本書における配信とは、Webにおいて（主にHTTPで）コンテンツ*1をサーバーからクライアントに届けることを指します。本書では配信をより高速・安定・安全にすることを配信最適化と定義します。配信を実現するさまざまな手段を配信技術、配信に関連するリバースプロキシ（Proxy）*2やCDN*3などを用いて構築した一連のシステムを配信システムとします。

配信の安定性や速度はユーザー体験に直結します。WebサイトやWebサービスを提供するうえで、避けては通れない領域です。

配信技術はサーバー・クライアント間にわたります。単純にどこか1つの知識があればよくできるものではありません。全体像を把握し、適切にアプリケーションを設定し、インフラ構成を組み、キャッシュを行い、場合によってはCDNなどの外部サービスを使うなど対応は多岐にわたります。

とはいえ、現段階では何のこともやらせましょう。ごく単純な構成で配信を意識して改善すると、どのように役に立つかを考えてみましょう。素朴なWebサイトやブログを立ち上げる際にはレンタルサーバーなどにサイトを設置することが多いです。

サーバーはインターネットにつながっており、スマートフォンやPCなどの**クライアント**はインターネットを経由してWebサイトを閲覧しに来ます。

*1 本書では、サーバーのローカルファイルをコンテンツリソース、サーバーのコード上で生成されたいわゆる動的なWebページなどとコンテンツリソースを合わせたものをコンテンツと定義します。なおリソースと単体で示したときは、CPUやストレージなどの資源を示します。

*2 プロキシサーバーの一種。本書では基本的にProxyと記載。Apache HTTP ServerなどWebサーバーや、アプリケーションサーバーの前段（クライアント側）に立て、クライアントからのリクエストをまとめてWebサーバーやアプリケーションサーバーに投げます。クライアントからのリクエストを整理したり、アプリケーションサーバーの出力内容をキャッシュしたりします。詳細は4章で解説します。

*3 Content Delivery Networkの略称。Webのコンテンツ配信をより安定化、より高速化するためのサービス。代表的なサービスにAkamaiやFastlyなどがある。

HTTPヘッダ・設定とコンテンツの見直し

配信に問題を見出したとき、おそらく最初に思い浮かぶ対策はCDN導入やサーバー増強でしょう。

ところが、少なくないサイトで、これらの対策にほとんど効果がないことがあります。このとき、問題はCDN導入やサーバー強化以前にあります。

- ・ 標準仕様の誤解から不用意な設定を行っていることでうまくローカルキャッシュが使えていない
- ・ 小さく表示するためのサムネイルなのに明らかに過大な画像が設定されている*1
- ・ ……

こんな事例を数多く見てきました。CDN導入や大規模な構成変更の手を付ける前に、改善すべきポイントは数多くあります。こういった改善を一つ一つ積み重ねることがサービスの快適性、安定性やコスト削減につながります。

本章では、こういった**大幅な構成変更を行わずにできる効率化**を解説していきます。

配信を行う環境はさまざまですが、総転送量の削減が利益に大きく影響する、という点はおおよそ共通しています。たとえばAWSをはじめとするクラウドサービスでは、転送量ベースの課金を採用しているものがあります*2。無計画に配信していると、ここが馬鹿にならない金額になります。

転送量（総転送量）は次のように求められます。

$$\text{総転送量} = \text{リクエスト数} \times \text{平均コンテンツサイズ}$$

本章では、次の2点を軸に、総転送量の削減を目指します。

- ・ 標準仕様にとったHTTPヘッダの設定を行い、ローカルキャッシュをう

*1 筆者はサムネイルで8K（7680×4320）ものサイズの画像と遭遇したことがあります。

*2 サービスによってはリクエスト数課金などもある。

キャッシュによる負荷対策

前章で、クライアントに保存されるローカルキャッシュを有効に使えるようになりました。これで負荷対策は完璧でしょうか？

実際に起こりうるケースで考えてみましょう。突然 SNS でサービスが話題になってユーザーが増えたらどうでしょう？ 2章で触れた通り、ローカルキャッシュはそのクライアントのみで再利用されます。つまり、すでにサイトを見ているクライアントには有効です。

ところがユーザーが増えるということは、新規の、ローカルキャッシュを持っていないクライアントからのリクエストが増えるということです。当然、ユーザーが増えれば増えるほどオリジンへのリクエストが増え、直撃する負荷は上昇します。ここまで主に解説してきたクライアント側のキャッシュだけでは、ユーザー数の単純な増加にはどうやら対応できなさそうです。

ここで役に立つのが Proxy や CDN で行う、経路上のキャッシュです。

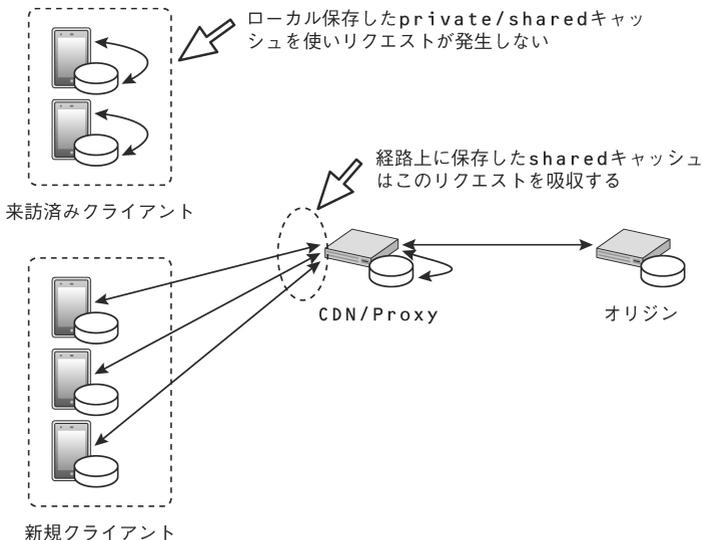


図 4.1 ローカルと経路上のキャッシュが効くところの違い

図のように新規クライアントからのリクエストも Proxy/CDN が経路上のキャッシュを持っていれば、そこからレスポンスします。持っていないと、オリジン

CDNを活用する

全世界のユーザーに向けて配信を行いたい、トラフィックが非常に大きい配信を行いたいという要件は、Webサービスを運営していれば当然のように出てきます。サイトが成長して、トラフィックの伸びに対して、既存構成だと耐えきれないので何とかしたいということもあるでしょう。

その際に便利に使えるのが、ここまで何度も名前を出してきたCDN（Content Delivery Network）です。

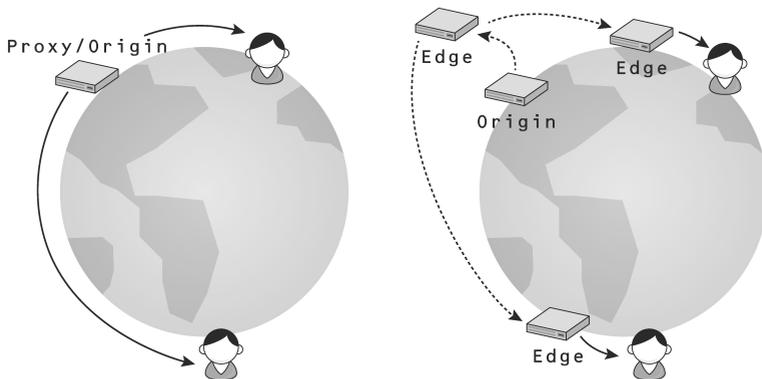


図6.1 CDNのエッジはクライアントの近くにある

もっともわかりやすいCDNのメリットは、配信の高速化、安定化でしょう。

CDNは世界中に高品質のネットワーク、エッジサーバー（クライアントと近い位置にあるサーバー）を有します。そのため、クライアントに地理的に近い位置から、高速かつ安定した配信が可能となります。

CDNをうまく導入すれば、全世界のエッジから、世界中のユーザーに向けて配信がより高速にできます。豊富な回線容量を持つため、大トラフィックもさばけます。

しかし、CDNは入れれば自動的になんとなく配信をいい感じにしてくれるものではありません。さまざまな使い方と注意点があるため、本章では詳細に解説します。

自作CDN

(DIY-CDN)

自作CDN (DIY-CDN) や自社配信 (自作CDNの一部) と外部のCDNを組み合わせるケースについて紹介しました (6.1.2 参照)。

自前のCDN (自社CDN) を採用する会社の代表例にNetflixがあります*1。莫大なトラフィックが常時ある、配信がコアビジネスであることから、外部のCDNサービスを利用するよりもコストやカスタマイズ性の面でメリットがあるでしょう。

Netflixのような大規模配信ではなくとも、ほかにも自作CDNがフィットするパターンもあります。そもそも低予算でCDNが使えない、コストを抑えたい、ハイブリッド構築したいパターンです。

本章ではそれぞれのパターンと、実際に低予算で自作CDNをつくるポイントを解説します。

7.1 なぜCDNをつくるのか

CDNを自作するケースは大別すると次の3つです。

- ・ 超大規模サービス。非常に大きなトラフィックや高度なカスタマイズの必要性から、自社構築にコストメリットがある (Netflix など)
- ・ 低予算。CDNを利用する資金がない
- ・ ハイブリッド構成。CDNと自社配信 (自作CDN) のハイブリッド構成によるメリットがある
 - 静的コンテンツのオフロードするためにCDNを使う
 - 定期的に流量があり、自社で低コストで回線調達が可能なため、ベースロードを自社配信として溢れた部分をCDNにオフロードする (DMMなど*2)
 - コンテンツを一貫させるために自社配信とCDNの多段構成にする

ほかにもコンテンツの性質上CDNが使えないなど、いくつか例外的なパターンもありえますが、割愛します。上述の3つの中で、多くの読者にとって現実的な

*1 Netflix OpenConnect というサービスを自前で運用。 https://openconnect.netflix.com/ja_jp/

*2 ちなみにAppleも自社でCDNを持っており、ハイブリッド構成となっています。 <https://arxiv.org/abs/1810.02978>